Interactive Authentication

Deepak Maram
Cornell Tech and *IC3*
sm2686@cornell.edu

Mahimna Kelkar
Cornell Tech and *IC3*
mahimna@cs.cornell.edu

Ittay Eyal
Technion and *IC3*
ittay@technion.ac.il

***Abstract*—Authentication is the first, crucial step in securing digital assets like cryptocurrencies and online services like banking and social networks. It relies on principals maintaining exclusive access to credentials like cryptographic signing keys, passwords, and physical devices. But both individuals and organizations struggle to manage their credentials, resulting in loss of assets and identity theft. Multi-factor authentication improves security, but its analysis and design are mostly limited to *one-shot mechanisms*, which decide immediately.**

In this work, we study mechanisms with back-and-forth *interaction* with the principals. For example, a user receives an email notification about sending money from her bank account and is given a period of time to abort the operation.

We formally define *the authentication problem*, where an *authentication mechanism* interacts with a user and an attacker and tries to identify the user. A mechanism's success depends on the scenario – whether the user / attacker know the different credentials; each credential can be safe, lost, leaked, or stolen. The *profile* of a mechanism is the set of all scenarios in which it succeeds. Thus, we have a partial order on mechanisms, defined by the subset relation on their profiles.

We find an upper bound on the profile size and discover three types of n-credential mechanisms (for any n) that are *maximally secure*, meeting this bound. We show these are all the unique maximal mechanisms for $n \leq 3$.

We show the efficacy of our model by analyzing existing mechanisms, both theoretical and deployed in widely-used systems, and make concrete improvement proposals. We demonstrate the practicality of our mechanisms by implementing a maximally-secure cryptocurrency wallet.

I. Introduction

Authentication plays a critical role in safeguarding online services and digital assets. An *authentication mechanism* binds a user's physical identity to a digital identity [1], [2]. It relies on the user's exclusive access to *credentials* like a password, a one-time PIN (OTP), or a cryptographic signing key. But credential management is challenging [3], [4], [5]: identity theft in the US is rampant [6]; an estimated 20% of Bitcoin have disappeared as a result of key loss [7]; and \$600M in cryptocurrency were stolen recently due to a single company's key mismanagement [8].

Some practical mechanisms employ *interaction*. For example, the Argent cryptocurrency wallet [9] uses multiple cryptographic signing keys as credentials. In a possible configuration, any 1-out-of-3 credentials can initiate a change of credentials, but the change only happens after two days, during which the operation can be cancelled with any 2-out-of-3 credentials. In online banking, some banks (e.g., [10]) introduce an artificial delay period before transferring funds; they notify the client and allow her to abort an erroneous transaction during this period. Government agencies (e.g., [11]) send a mail notice about reversible account activity, allowing victims to revert in case of identity theft attempts [6].

But despite being used in practice, to the best of our knowledge, interactive authentication has never been formally studied and its importance is therefore under-appreciated. Prior works (§II) have focused on proposing frameworks that only model non-interactive mechanisms [12], [13], or consider specific interactive mechanisms like the ones mentioned above [9], [14], which we show to be sub-optimal. Multi-factor authentication is typically defined as requiring multiple credentials [15], i.e., combine credentials using a conjunction (AND) operator, without taking advantage of interactivity. Similarly, multi-sig or threshold mechanisms (k-out-of-n keys) are popular in the cryptocurrency industry [16], [17], [18], but are non-interactive.

In this work, we formally study interactive authentication mechanisms. We first define the *authentication problem* (§III), consisting of a *mechanism* and two players: a *user* and an *attacker* that stands for all entities trying to authenticate as the user. The mechanism is a *deterministic finite automaton* using $n \geq 1$ credentials to identify the user. Each credential can be in one of four states [12]: *safe* (only the user has it), *stolen* (only the attacker has it), *leaked* (both have it) or *lost* (neither has it). The states of all credentials define a *scenario*, with a total of 4^n scenarios possible.

The players send to the mechanism messages carrying proofs of the credentials they have, and eventually the mechanism *decides* which of them is the user. A mechanism *succeeds* in a particular scenario if it is correct irrespective of what the attacker does; otherwise it *fails*. When the communication channels are *synchronous*, the mechanism can rely on an interactive message exchange with the players.

To evaluate mechanisms, we define the *security profile* of a mechanism to be the set of all scenarios in which it is successful. For example, with two credentials denoted by c_1 and c_2, consider the OR-mechanism where either c_1 or c_2 can be used to authenticate, i.e., $c_1 \vee c_2$. Its profile has three scenarios: (c_1 and c_2 safe), (c_1 safe, c_2 lost) and (c_1 lost, c_2 safe). The mechanism succeeds in the latter two because the user knows enough credentials to authenticate and the attacker does not. Note that the mechanism fails in scenarios where both players have 1–2 credentials, e.g., (c_1 stolen, c_2 safe), because we conservatively assume that the adversary controls the order in which messages are delivered to the mechanism.

The security profile defines a relation between any two mechanisms M_1 and M_2 using the same number of credentials: M_1 is *better* than M_2 if the profile of M_1 is a superset of the profile of M_2. M_1 is *equivalent* to M_2 if they have the same profile. Otherwise, M_1 and M_2 are incomparable. A mechanism is *maximally secure* or simply *maximal* if no other mechanism is better.

By proving constraints on the profile sets any mechanism could achieve, we bound the profile size (§IV). We find that with n-credentials, no mechanism succeeds in more than $P(n) = \frac{4^n - 2^n}{2}$ scenarios. For example, $P(2) = 6$, that is, a 2-credential mechanism can win in at most 6 scenarios. Our use of a generic computational model for mechanisms, namely automatons, allows us to prove this impossibility result.

The bound is tight as we discover several maximal mechanisms meeting it (§V). All our maximal mechanisms function as follows: Either player can initiate the mechanism, at which point the mechanism starts a timer. Until the time runs out, each player can submit messages to the mechanism, carrying one or more credentials. Finally, the mechanism decides on the winner. We refer to mechanisms (maximal or otherwise) following the above structure as *bounded-delay mechanisms*. They only differ in the way a winner is decided.

We identify three sufficient properties for a bounded-delay mechanism to be maximal: (1) *ID-Agnostic*: the mechanism only decides based on messages, not other information; (2) *Knowledge-rewarding*: if a player submits a credential set that is a superset of its counterpart's set, then the mechanism decides on the former; and (3) *Transitive-knowledge-rewarding*: submitting additional credentials cannot lead to a worse outcome for the submitting player.

We discover three classes of bounded-delay mechanisms that satisfy these properties. The first two are fairly intuitive: *Majority mechanisms* choose whoever submits the most credentials, with some tie-breaking function. *Priority mechanisms* use a priority vector defined over the n-credentials, and the winner is the player that submits a unique higher priority credential. For example, consider the priority vector $[c_2, c_3, c_1]$: if player P1 submits $\{c_1, c_2\}$ and player P2 submits $\{c_2, c_3\}$, then P2 wins because c_3 has a higher priority than c_1.

The third class is similar to the priority mechanisms but with an exception: If each of the two players submit one of the last two credentials, then the priority is reversed. Using the priority vector above, if P1 submits $\{c_1\}$ and P2 submits $\{c_3\}$, then P1 wins. This is the only exception and we use the priority rule otherwise, e.g., if P1 submits $\{c_1, c_2\}$ and P2 submits $\{c_2, c_3\}$, then P2 wins like before.

We can now illustrate the advantage of interactivity by going back to the 2-credential setting. Consider the 2-credential priority mechanism defined by the vector $[c_1, c_2]$. Its profile contains (c_1 and c_2 safe); (c_1 safe, c_2 leaked or lost or stolen) because only the user knows the highest-priority credential c_1; and (c_1 leaked or lost, c_2 safe) because c_2 is safe and c_1 is known to either both the players or to neither. In total, the size of the profile is six, equal to the bound $P(2)$, with three more scenarios than the OR-mechanism described above.

The three mechanism classes cover the *complete set* of n-credential maximal mechanisms for $n \leq 3$ (§VI), i.e., *any* mechanism *must be either worse than or equivalent* to a mechanism in the three classes.

We show the efficacy of our model by analyzing some existing interactive mechanisms (§VII), including the popular cryptocurrency wallet Argent [9] used by over a million users [19]. Argent lets the user select m key guardians (e.g., friends) and keeps one key on the user's phone for a total of $n = m + 1$ credentials. It provides functionality to transfer funds, add or delete any credential, and lock the wallet in case of a suspected key fault.

We find that Argent's mechanism has 5 scenarios if $n = 2$ and 22 scenarios if $n = 3$ when used according to the Argent documentation. We also propose a more involved user-side technique allowing it to reach 6 and 24 scenarios, respectively. So, even with our technique, Argent is only maximal when $n = 2$ but not for greater n, e.g., 3-credential Argent is worse than the priority mechanism ($P(3) = 28$ scenarios).

We demonstrate our mechanisms are practical with a priority mechanism implementation in Solidity (Appendix A). Its most expensive function costs \$7.6 in fees, compared to \$5.4 for Argent (at today's rates; see §VII-A).

In summary, our main contributions are:

1) A definition of the authentication problem, in particular under synchrony,
2) a metric of the security level of an authentication mechanism, namely security profiles,
3) a tight upper bound on the profile size of any mechanism,
4) novel n-credential mechanisms that are maximally secure,
5) the complete sets of maximal mechanisms for $n \leq 3$,
6) security analysis of deployed and theoretical mechanisms with concrete improvement proposals, and
7) a Solidity implementation of a maximally-secure wallet.

We conclude (§VIII) with practical insights and a brief review of open questions.

II. Related Work

To the best of our knowledge, previous work did not provide a general definition of the authentication problem or analyzed the interactive authentication design space.

The closest work is Eyal's cryptocurrency wallet design [12], which investigates what the best mechanism is, but in a restrictive setting with only boolean formulae, i.e., one-shot mechanisms. Our maximal mechanisms have larger security profiles. Hammann et al. [13] use a similarly restrictive model and their focus is orthogonal to ours, namely, to uncover vulnerabilities arising from the links between different mechanisms a user uses.

There is substantial interest in analyzing the security of multi-factor authentication mechanisms. For example, Jacomme et al. [20] and Barbosa et al. [21] analyze Google's 2FA and Fido's U2F. But they do not propose generic frameworks; moreover, these mechanisms are also one-shot.

Elaborate authentication mechanisms are common for cryptocurrency assets [22], [16], [18], [23]; some of which even use interactive schemes like Argent [9], SmartCustody [24] and CoinVault [25] (which was part of the inspiration for this work). This demonstrates the practicality of interactive schemes in this context, but also the need for rigor, as we demonstrate by analyzing and proposing improvements to the popular Argent in §VII-A.

Paralysis Proofs [14] leverages interactivity to deal with credential loss; similar ideas were recently proposed in the Bitcoin community [24]. We discuss this approach in §VII-C and show that our mechanisms achieve larger security profiles.

Vaults [26] (the first interactive cryptocurrency wallet design to our knowledge) and KELP [27] leverage interactivity to deal with key leakage and loss, respectively. But their models are distinct from ours: Vaults [26] treats the case where neither player can withdraw funds as success, since it removes the motivation for an attack; we conservatively treat this outcome as a failure. KELP [27] considers partial knowledge – a user knows about a lost key before anyone else, whereas we only consider complete knowledge. Both works consider specific mechanisms whereas we find bounds and maximal mechanisms.

Work on authorization dealt with the implementation of security policies [28], [29], [30]. In some contexts, our mechanisms can be implemented using those solutions.

III. Model

We formalize the authentication problem. We describe an execution, its participants and communication (§III-A), credentials (§III-B), the automaton model for authentication mechanisms (§III-C), and player strategies (§III-D), all summarized in Algorithm 1. Finally, we define security profiles, with which we evaluate mechanisms (§III-E).

A. Execution: participants, time and network model

The system comprises an *authentication mechanism* M and two players, a *user* U and an *attacker* A.

An execution begins with the environment (an entity we use to orchestrate a real-world situation) assigning distinct *player identifiers* to the user and to the attacker from the set $\mathcal{P} = \{0, 1\}$. It picks $\gamma \in \mathcal{P}$ and assigns γ to the user and $(1-\gamma)$ to the attacker, e.g., if $\gamma = 0$, then the user is player 0 and the attacker is player 1. The identifiers are akin to cookies used to identify a website visitor during a single session. We refer to γ as the *environment's strategy*.

Time progresses in discrete steps. During the execution, both parties interact with the mechanism by sending and receiving messages. Communication channels are reliable and synchronous – if a message is sent in time step a, it reaches the recipient in step $a + 1$.

The attacker controls the ordering of messages within a time step. That is, if both user and attacker send messages to the mechanism at the same time step, the attacker can choose which of the two is received first.

The mechanism can decide either 0 or 1 to end an execution, and the player with this identifier *wins*. This is a one-time irrevocable operation corresponding to, say, allowing someone to withdraw money out of a bank account.

B. Credentials, scenarios and messages

The system contains a set of n credentials, $\mathcal{C}_{\mathsf{all}} = \{c_1, c_2, \dots, c_n\}$. Each credential is in one of four states [12]: (1) *Safe:* Only U has it, (2) *Lost:* No one has it, (3) *Leaked:* Both U and A have it, or (4) *Stolen:* Only A has it. Denote the state of credential c_i by σ_i, so for all $1 \leq i \leq n : \sigma_i \in \{\mathsf{safe}, \mathsf{lost}, \mathsf{leaked}, \mathsf{stolen}\}$. A *scenario* σ is a vector of the n credential states and $\Sigma = \{\mathsf{safe}, \mathsf{lost}, \mathsf{leaked}, \mathsf{stolen}\}^n$ is the set of all scenarios, i.e. $\sigma \in \Sigma$.

The *credential set* of the user (resp., attacker) contains all the credentials available to that player, denoted $C^U_\sigma = \{c_i | \sigma_i \in \{\mathsf{safe}, \mathsf{leaked}\}\}$ (resp., $C^A_\sigma = \{c_i | \sigma_i \in \{\mathsf{leaked}, \mathsf{stolen}\}\}$).

At the start of an execution, the environment picks a scenario $\sigma \in \Sigma$ and initializes U and A with the credential sets C^U_σ and C^A_σ, respectively. The scenario σ is *common knowledge*, i.e., the state of all n credentials is known to both parties.

Each message sent by U or A consists of a player identifier $p \in \mathcal{P}$ and a set of credentials C. A message can only carry a subset of the credentials available to the sender, i.e., a user (resp., attacker) message can carry a set of credentials C s.t. $C \subseteq C^U_\sigma$ (resp., $C \subseteq C^A_\sigma$). To avoid credential leakage by listening to the channel, a message should actually contain encrypted credentials or zero knowledge proofs of credential knowledge, e.g., signatures for private keys etc. We avoid this detail for simplicity.

Although an attacker cannot include a credential not assigned to it, we do not preclude the attacker from forging credentials. Forgery is modeled as leakage or stealing.

C. Authentication mechanism

The mechanism M is a deterministic one-clock timed automaton [31], defined by the following elements:

States: The automaton has a finite set $\mathcal{T}$ of *states*. It includes a special *starting state* I_{init} and two disjoint sets of *final states*, $\mathcal{T}^{\mathsf{fin}}_0$ and $\mathcal{T}^{\mathsf{fin}}_1$. The execution ends as soon as the automaton reaches a final state, i.e., if $s \in \mathcal{T}^{\mathsf{fin}}_i$, then the winner is the player with identifier i.

Clock: A *clock* has some value $t \in \mathbb{Z}_{\geq 0}$, initialized with $t = 0$, and incremented by 1 in each step. The automaton can reset the clock back to its initial state ($t = 0$).

A pair (s, v) of an automaton state s and a clock state t is called an *extended state*. At the start of an execution, we have $s \leftarrow I_{\mathsf{init}}$ and $t \leftarrow 0$. Note that we refer just to the automaton state s when using the word state.

Transitions: An automaton transition is a 6-tuple consisting of a source and a destination state, three types of guards and a clock reset bit, as follows.

The *Clock Reset* $\mathcal{R} = \{\mathsf{True}, \mathsf{False}\}$ represents the reset of the clock state or no reset. Resetting means that the clock is set back to its initial state ($t \leftarrow 0$) upon taking the transition.

Transitions take place on message arrival. A guard g is a constraint specifying that the message must meet a condition for the transition to take place. A guard with the value $\perp$ indicates no constraint. There are three types of guards:

A *Player guard* means the message must be sent by a certain player ID. The set of player identifier guards is $\mathcal{G}_{\mathsf{id}} = \mathcal{P} \cup \{\perp\}$. If a player $p \in \mathcal{P}$ satisfies the guard g^{plr} we denote $p \vdash g^{plr}$. For example, $0 \vdash g^{plr}$ only if $g^{plr} \in \{0, \perp\}$.

A *Credential guard* means that the message must carry certain credentials. The set of credential guards $\mathcal{G}_c$ includes all non-constant monotone boolean formulae of the availability of the credentials in the set $\mathcal{C}_{\mathsf{all}}$ and $\perp$. For example, if $n = 2$, then $\mathcal{G}_c = \{c_1, c_2, c_1 \wedge c_2, c_1 \vee c_2, \perp\}$. A credential set C satisfies a credential guard g^{cd}, denoted $C \vdash g^{cd}$, if the credentials in C satisfy the boolean formula g^{cd}. For example, if $g^{cd} = c_1 \vee c_2$ and $C = \{c_1\}$, then $C \vdash g^{cd}$.

A *Clock guard* is a condition on the clock state t expressed through a comparison operator $\sim \in \{<, \leq, =, \geq, >\}$ and a natural number $l \in \mathbb{N}$, i.e., $(t \sim l)$. The set of all clock guards (including $\perp$) is $\mathcal{G}_t$. For example, a guard $g^{clk} = (t < 5)$ specifies that the clock state t must be less than 5. Denote by $t \vdash g^{clk}$ that the clock state t satisfies the guard g^{clk}.

In summary, $\mathcal{D} \subseteq \mathcal{T} \times \mathcal{G}_{\mathsf{id}} \times \mathcal{G}_c \times \mathcal{G}_t \times \mathcal{R} \times \mathcal{T}$ is the set of transitions of M. (The source state must be a non-final state, and the target state not the beginning state.) And an automaton M is a 6-tuple given by $M = (\mathcal{C}_{\mathsf{all}}, \mathcal{T}, clock, \mathcal{D}, \mathcal{T}_0^{\mathsf{fin}}, \mathcal{T}_1^{\mathsf{fin}})$.

If the current state is s and the automaton receives a message satisfying all the guards of an outgoing transition of s, then the destination state of the transition becomes the new state. There can exist at most one such transition because we consider a deterministic automaton, i.e., we require: For all states $s \in \mathcal{T}$, all clock states $t \in \mathbb{Z}_{\geq 0}$, all sets of credentials $C \subseteq \mathcal{C}_{\mathsf{all}}$, and all player identifiers $p \in \mathcal{P}$, there exists at most one transition $(s, g^{plr}, g^{cd}, g^{clk}, r, s') \in \mathcal{D}$ such that all the guards are satisfied, i.e., $p \vdash g^{plr}$, $C \vdash g^{cd}$ and $t \vdash g^{clk}$.

The automaton notifies both parties on each state change, including if there are multiple transitions in a single time step.

Note 1. *We use a clock in automatons for illustrative reasons. It can be removed to construct a deterministic finite automaton using standard techniques (see region automata [31]).*

D. Player strategies and mechanism success

The *user strategy* S^U specifies which messages the user sends in any given extended mechanism state:

Definition 1 (Messaging strategy)**.** *A messaging strategy for the user (resp., attacker) is a function of the extended state $e = (s, t)$, $S(e)$ that returns $\perp$ or an ordered list of credential sets $\{C_k\}$. A messaging strategy is* playable *by the user (resp., attacker) in a scenario σ if for all automaton extended states e and for all $C_k \in S(e) : C_k \subseteq C_\sigma^U$ (resp., $C_k \subseteq C_\sigma^A$).*

A player adopting a messaging strategy S means that if the current extended state is e, then the player sends one message with each of the credential sets in $S(e)$ in the next time step.

The attacker's strategy S^A consists of two components: a messaging strategy S^A_{msg} and an *ordering strategy* S^A_{ord}, i.e., $S^A = (S^A_{\mathsf{msg}}, S^A_{\mathsf{ord}})$. The latter captures how an attacker orders messages that arrive in the same time step. More formally, S^A_{ord} takes both the user's and attacker's messages as inputs and outputs some permutation of them. Two examples of ordering strategies are the *attacker-first* strategy Ord^A, where the attacker orders its own messages before the user's, and the *user-first* strategy Ord^U, where the user's messages are first. Unless specified otherwise, the attacker employs Ord^A.

Algorithm 1 Execution with players U and A and mechanism $M = (\mathcal{C}_{\mathsf{all}}, \mathcal{T}, clock, \mathcal{D}, \mathcal{T}_0^{\mathsf{fin}}, \mathcal{T}_1^{\mathsf{fin}})$

```
Init: The environment chooses a scenario $\sigma \in \Sigma$ and $\gamma \in \{0,1\}$. It initializes U (resp., A) with the player identifier $\gamma$ (resp., $(1-\gamma)$) and the set of credentials $C_\sigma^U$ (resp., $C_\sigma^A$).
$s \leftarrow I_{\mathsf{init}}$                                  ▷ Automaton state
$t \leftarrow 0$                                        ▷ Automaton clock
Execution:
  repeat                                 ▷ Execute loop once per time step
    $Y_{\mathrm{U}} \leftarrow \{(\gamma, C) | C \in S^U((s,t))\}$          ▷ U sends messages
    $Y_{\mathrm{A}} \leftarrow \{((1-\gamma), C) | C \in S^A_{\mathsf{msg}}((s,t))\}$        ▷ A sends messages
    $\{(p_i, C_i)\}_{i=1}^{|Y_{\mathrm{U}} \cup Y_{\mathrm{A}}|} \leftarrow S^A_{\mathsf{ord}}(Y_{\mathrm{U}} \cup Y_{\mathrm{A}})$        ▷ A orders messages
    for $i = 1, 2, \ldots, |Y_{\mathrm{U}} \cup Y_{\mathrm{A}}|$ do ▷ $M$ processes messages
      if $\exists (s', g^{plr}, g^{cd}, g^{clk}, r, s'') \in \mathcal{D}$, $s = s'$,
                  $p_i \vdash g^{plr}$, $C_i \vdash g^{cd}$, and $t \vdash g^{clk}$ then
        $s \leftarrow s''$                      ▷ Update automaton state
        if $r = \mathsf{True}$ then $t \leftarrow -1$.               ▷ Reset clock
        if $s \in \mathcal{T}_0^{\mathsf{fin}}$ then return 0
        if $s \in \mathcal{T}_1^{\mathsf{fin}}$ then return 1
    $t \leftarrow t + 1$                                    ▷ Advance clock
```

An *execution* E is fully defined by an automaton M, a scenario σ, player strategies S^U and S^A, and the environment's strategy γ, i.e., $E = (M, \sigma, S^U, S^A, \gamma)$. The details of how an execution unfolds are specified in Algorithm 1.

The user wins only if the mechanism chooses its identifier:

Definition 2 (Execution winner)**.** *Given an execution $E = (M, \sigma, S^U, S^A, \gamma)$, the* winner *of an execution is the player decided by the automaton M, or the attacker A if M never decides. We denote $Win^{\mathsf{exec}}(E) \in \{U, A\}$ accordingly.*

The winner of two executions played with the same player strategies can differ due to a change in environment's strategy.

A mechanism M *succeeds* in a scenario σ, denoted $Suc(M, \sigma)$, if the user can win irrespective of the attacker's strategy:

Definition 3 (Success)**.** *Given a mechanism M and a scenario σ, we say that the mechanism is* successful,*denoted $Suc(M, \sigma)$, if the user consistently wins against any attacker's strategy and environment's strategy, i.e., $\exists S^U_{\mathsf{win}} : \forall S^A, \forall \gamma,\ E = (M, \sigma, S^U_{\mathsf{win}}, S^A, \gamma)$ and $Win^{\mathsf{exec}}(E) = U$. Otherwise, the mechanism fails, denoted $\neg Suc(M, \sigma)$.*

Let $X = (M, \sigma)$ denote a mechanism and a scenario, and call X an *extended scenario*. We say $Win_X(S^U, S^A) = \mathrm{U}$ if the strategy S^U wins for the user in all executions where the attacker employs S^A, i.e., independent of the environment's

Notation	Description
U, A	User, Attacker
$\mathcal{C}_{\mathsf{all}} = \{c_1, \ldots, c_n\}$	The n credentials
σ, $\overline{\sigma}$	Scenario and its complement
$\sigma_i \in \{\mathsf{safe}, \mathsf{lost}, \mathsf{leaked}, \mathsf{stolen}\}$	State of c_i
C^U_σ / C^A_σ	Credentials of user / attacker in σ
$\Sigma_{\mathsf{ws}}, \Sigma_{\mathsf{ns}}$	All with-safe, no-safe scenarios
$\Sigma_{\mathsf{wt}}, \Sigma_{\mathsf{nt}}$	All with-stolen, no-stolen scenarios
$\mathcal{P} = \{0, 1\}$	Player identifiers
$M = (\mathcal{C}_{\mathsf{all}}, \mathcal{T}, \mathit{clock}, \mathcal{D}, \mathcal{T}^{\mathsf{fin}}_0, \mathcal{T}^{\mathsf{fin}}_1)$	Automaton / Mechanism
$\mathcal{T}^{\mathsf{fin}}_0 / \mathcal{T}^{\mathsf{fin}}_1$	Final states where 0 / 1 wins
$\mathcal{G}_c, \mathcal{G}_t, \mathcal{G}_{\mathsf{id}}$	Credential, clock and player guards
$C \vdash g^{cd}$	Credential guard
clock, t	The clock and its state
$g^{clk} := t \sim l,\ t \vdash g^{clk}$	Clock guard
$r \in \{\mathsf{True}, \mathsf{False}\}$	Clock reset
$(s, g^{plr}, g^{cd}, g^{clk}, r, s') \in \mathcal{D}$	A 6-tuple edge in the automaton
$\mathit{Suc}(M, \sigma)$	Is the mechanism successful in σ?
S^U, S^A	Strategies of user and attacker
$\mathit{Win}^{\mathsf{exec}}(E)$	Winner of an execution
$\mathit{Win}_\sigma(S^U, S^A)$	Winner under specified strategies
$\mathit{prof}(M)$	Security profile of the scheme M
$P(n)$	Bound on the profile size
$\mathcal{M}_n$	All n-credential mechanisms
$\mathcal{O}_n$	Complete set of mechanisms

TABLE I: Notation

strategy. A strategy S^U_{win} is a winning user strategy if it wins against any attacker strategy, i.e., $\forall S^A : \mathit{Win}_X(S^U_{\mathsf{win}}, S^A) = \text{U}$.

On the other hand, $\mathit{Win}_X(S^U, S^A) = \text{A}$ means that S^A wins for the attacker in some execution where the user employs S^U, i.e., with some player identifier allocation. And S^A_{win} is a winning strategy for the attacker if $\forall S^U : \mathit{Win}_X(S^U, S^A_{\mathsf{win}}) = \text{A}$.

We conclude this section with a few remarks.

Parallel executions: Multiple executions can run in parallel at the same time although we do not explicitly model it.

State updates: Recall that we assume that any state change is immediately informed to both the user and attacker. This is both a strict and conservative assumption: strict (resp., conservative) because the user (resp., attacker) knows all the state updates, including any authentication attempts by the attacker (resp., user).

E. Mechanism profiles

Having defined mechanism success, we can now evaluate and compare mechanisms. A mechanism's *profile* is a concise representation of its security level containing all the scenarios in which it succeeds.

Definition 4 (Profile)**.** *The profile of a mechanism M denoted by $\mathit{prof}(M)$ is the set of all scenarios where M succeeds, i.e., $\mathit{prof}(M) = \{\sigma | \sigma \in \Sigma \wedge \mathit{Suc}(M, \sigma)\}$.*

The profile can also be viewed as an n-dimensional matrix where n is the number of credentials. Each matrix cell represents a distinct scenario and the value in it is 1 or 0 if the scenario is in the profile or isn't, respectively (e.g., Fig. 1).

The profiles define a partial order on the set of *all n-credential mechanisms*, denoted by $\mathcal{M}_n$. Two mechanisms $M, M' \in \mathcal{M}_n$ are *equivalent*, denoted by $M \cong M'$, if one's profile can be obtained from the other's by permuting the credential set:

Definition 5 (Equivalence)**.** *Given a permutation $\pi : \{1, \ldots, n\} \to \{1, \ldots, n\}$ and a scenario σ, define the permuted scenario σ^π by $\sigma^\pi_i = \sigma_{\pi(i)}$. Then, $M \cong M'$ if $\exists \pi : \bigcup_{\sigma \in \mathit{prof}(M)} \sigma^\pi = \mathit{prof}(M')$.*

A mechanism M_1 is *better* than another mechanism M_2, denoted by $M_1 \succ M_2$, or M_2 is *worse* than M_1, if $\exists M_3 \cong M_2$ such that $\mathit{prof}(M_1) \supset \mathit{prof}(M_3)$. M_1 is *better than or equivalent to* M_2, denoted by $M_1 \succeq M_2$, if $M_1 \succ M_2$ or $M_1 \cong M_2$ ($\preceq$ defined analogously). And M_1 is *incomparable* to M_2, denoted by $M_1 \not\approx M_2$, if $M_1 \not\succeq M_2 \wedge M_1 \not\preceq M_2$.

We give an example to illustrate the relations. As we saw (§I), the profile of the 2-credential OR-mechanism $c_1 \vee c_2$ has three scenarios: (c_1 and c_2 safe), (c_1 safe, c_2 lost) and (c_1 lost, c_2 safe). Now consider the AND-mechanism $c_1 \wedge c_2$: its profile also has three scenarios, namely (c_1 and c_2 safe), (c_1 safe, c_2 leaked) and (c_1 leaked, c_2 safe). We can see that the two mechanisms are incomparable as neither is better than the other nor are they equivalent.

The relation naturally defines *maximal* mechanisms:

Definition 6 (Maximally secure mechanism)**.** *A mechanism $M \in \mathcal{M}_n$ is* maximally secure *or* maximal *if for all $M' \in \mathcal{M}_n$: $M' \preceq M$ or $M' \not\approx M$.*

IV. Profile size bound

We find an upper bound on the profile set size, i.e., given n, we find the maximum number of scenarios in which an n-credential mechanism succeeds. Before that, we define *runs* and then prove a few useful results.

A *run* $r = \{(p_1, C_1, a_1), \ldots, (p_z, C_z, a_z)\}$ tracks all the timestamped messages (a_i denotes the time step) received by the automaton during an execution, irrespective of whether they lead to a state change or not. First, we show that any two executions with the same run have the same winning identifier.

Observation 1. *If two executions E_1 and E_2 using the same mechanism M have the same finite run, then the winner of both executions has the same player identifier.*

If E_1 and E_2 share the same run and use the same automaton, then they will cause the same state changes because we are using a deterministic automaton, and thus have the same final state s. Based on whether $s \in \mathcal{T}^{\mathsf{fin}}_0$ or $s \in \mathcal{T}^{\mathsf{fin}}_1$, player 0 or 1, respectively, wins in both executions E_1 and E_2.

Next we prove that if the user wins in all executions where the attacker employs a specific strategy (i.e., independent of the environment's strategy), then switching the messaging strategies allows the attacker to win an execution.

Lemma 1. *Let there be a mechanism M, two extended scenarios $X_1 = (M, \sigma_1)$, $X_2 = (M, \sigma_2)$, and two messaging strategies S_1, S_2 such that S_1 is playable by U in σ_1 and* *by A in σ_2 and S_2 is playable by U in σ_2 and by A in σ_1. There exists a reordering strategy $\overline{S^A_{\text{ord}}}$ such that the two attacker strategies $S^A_1 = (S_2, Ord^A)$, $S^A_2 = (S_1, \overline{S^A_{\text{ord}}})$ satisfy $(Win_{X_1}(S_1, S^A_1) = U) \implies (Win_{X_2}(S_2, S^A_2) = A)$.*

Proof. $Win_{X_1}(S_1, S^A_1) =$ U means that the user wins in all executions where the attacker's strategy is S^A_1. Consider such an execution $E = (M, \sigma_1, S_1, (S_2, Ord^A), 0)$, i.e., the user is assigned 0 and wins. Denote the run of E by r.

Consider another execution $E' = (M, \sigma_2, S_2, (S_1, \overline{S^A_{\text{ord}}}), 1)$, i.e., the attacker is assigned 0 and $\overline{S^A_{\text{ord}}} \leftarrow Ord^U$ is the user-first ordering strategy. If the mechanism in execution E' does not decide, i.e., E' runs forever, then by definition, $Win^{\text{exec}}(E') =$ A, and we are done. Now say the mechanism in E' decides; let its run be r'. We now prove that the two runs are same, i.e., $r = r'$. First, note that in both executions, player 0 uses messaging strategy S_1 and player 1 uses messaging strategy S_2. Secondly, since the attacker's reordering strategy prioritizes the attacker in E_1 and the user in E_2, the messages sent by 1 are prioritized in both executions. Therefore, the messages the mechanism receives in all time steps are the same in both executions including the order of messages. Therefore $r = r'$. Since player 0 (user) wins E_1, by Observation 1, the attacker wins execution E_2. □

Now, in order to bound the number of successful scenarios, we identify scenarios where a mechanism cannot succeed. First, we define *complementary scenarios*, where the players' credential availability is inverted, and show that no mechanism succeeds in both a scenario and its complement.

Definition 7 (Complement scenario). *A scenario $\overline{\sigma}$ is the* complement scenario *of σ if for all i: $\overline{\sigma}_i = \textsf{stolen}$ if $\sigma_i = \textsf{safe}$, $\overline{\sigma}_i = \textsf{safe}$ if $\sigma_i = \textsf{stolen}$, and $\overline{\sigma}_i = \sigma_i$ otherwise.*

Lemma 2. *If a mechanism succeeds in a scenario σ, then it fails in its complement $\overline{\sigma}$: $Suc(M, \sigma) \implies \neg Suc(M, \overline{\sigma})$.*

The intuition behind the proof is that the attacker will be able to employ in $\overline{\sigma}$ the same messaging strategy that the user uses in σ. Because the user's strategy wins in all executions against any attacker strategy in σ, we show that the attacker's strategy wins in at least one execution of $\overline{\sigma}$.

Proof. Let S^U denote a winning user strategy in the extended scenario $X = (M, \sigma)$. Since S^U wins in all executions, we have $\forall S^A : Win_X(S^U, S^A) =$ U. We now produce a winning strategy $\bar{S}^A$ for the attacker in $\bar{X} = (M, \overline{\sigma})$ based on S^U. To do this, we need to show two things, namely, that S^U is a playable messaging strategy for the attacker in $\overline{\sigma}$, and that it succeeds in at least one execution.

The first follows in a straightforward way from the definition of a complement. It is easy to see that $C^U_\sigma = C^A_{\overline{\sigma}}$. Therefore, any user strategy, including S^U, in the scenario σ is a playable attacker messaging strategy in the complement scenario $\overline{\sigma}$.

Next we need to show that the attacker wins an execution with S^U in $\overline{\sigma}$. Recall that Ord^A denotes the attacker-first reordering strategy. Let the attacker's strategy be

$$\bar{S}^A = \{S^U, Ord^A\}.$$

We need to show that no user strategy succeeds in all executions of $\bar{X} = (M, \overline{\sigma})$, i.e., $\nexists \bar{S}^U : Win_{\bar{X}}(\bar{S}^U, \bar{S}^A) =$ U.

We prove by contradiction. Assume the existence of a winning user strategy $\bar{S}^U$, i.e.,

$$Win_{\bar{X}}(\bar{S}^U, \bar{S}^A) = \text{U}.$$

We will use the successful strategy $\bar{S}^U$ to devise a strategy S^A for the attacker in the original scenario.

Apply Lemma 1 by setting $\sigma_1 = \overline{\sigma}$, $\sigma_2 = \sigma$, $S_1 = \bar{S}^U$ and $S_2 = S^U$. A pre-condition for this lemma is that S_1 ($\bar{S}^U$) needs to be playable by the attacker in σ_2 (σ), which holds because $C^U_{\overline{\sigma}} = C^A_\sigma$. Lemma 1 guarantees the existence of an ordering strategy S^A_{ord} such that

$$Win_{\bar{X}}(\bar{S}^U, \bar{S}^A) = \text{U} \implies Win_X(S^U, S^A) = \text{A}.$$

We thus found an attacker strategy (S^A) that wins an execution against the user winning strategy S^U in the original scenario σ—a contradiction. Therefore, the assumption was wrong and there does not exist a winning strategy for $\overline{\sigma}$. □

After dealing with complementary scenarios, we show that if no credential is safe, then no mechanism succeeds. We call such scenarios *bad*.

Lemma 3. *Given a bad scenario $\sigma^{\text{bad}} \in \{\textsf{lost}, \textsf{leaked}, \textsf{stolen}\}^n$, where all credentials are unsafe, $\nexists M \in \mathcal{M}_n : Suc(M, \sigma^{\text{bad}})$.*

We prove this by contradiction, i.e., say the user wins in σ^{bad}. Using the winning strategy of the user, we show the existence of a winning strategy for the attacker, thus arriving at a contradiction.

Proof. By contradiction, assume there exists a mechanism M that succeeds in a bad scenario, i.e., $Suc(M, \sigma^{\text{bad}})$. Denote the extended scenario $X = (M, \sigma^{\text{bad}})$. Let the winning user strategy be S^U, so for all $S^A, Win_X(S^U, S^A) =$ U. Consider the attacker strategy $S^*_1 = \{S^U, Ord^A\}$. We have $Win_X(S^U, S^*_1) =$ U.

Apply Lemma 1 by setting $\sigma_1 = \sigma$, $\sigma_2 = \sigma$, $S_1 = S^U$, and $S_2 = S^U$. A pre-condition for this lemma is that S^U be playable by both user and attacker in σ. This is true because in any scenario $\sigma \in \sigma^{\text{bad}}$, $C^U_\sigma \subseteq C^A_\sigma$. Consequently, for any set of credentials C, if $C \subseteq C^U_\sigma$ then $C \subseteq C^A_\sigma$. So S^U is playable for the attacker.

We have $Win_X(S^U, S^*_1) =$ U, therefore due to Lemma 1, there exists some S^A_{ord} such that if $S^*_2 = \{S^U, S^A_{\text{ord}}\}$ then $Win_X(S^U, S^*_2) =$ A. This completes the proof because S^*_2 wins against S^U, contradicting the assumption that S^U is a winning user strategy. Therefore the assumption was wrong and there does not exist a winning user strategy in a bad scenario. □

We can now bound the size of feasible security profiles.

Theorem 1. *The maximum number of scenarios an n-credential mechanism succeeds is*

$$P(n) = (4^n - 2^n)/2 \ , \tag{1}$$

i.e., $\forall M \in \mathcal{M}_n : |\mathit{prof}(M)| \leq P(n)$.

For proving, we first identify four subsets of scenarios, which will also be useful later. Define a *with-safe scenario* (resp., *with-stolen scenario*) as a scenario in which at least one credential's state is safe (resp., stolen). Denote the set of all with-safe (resp., with-stolen) scenarios by Σ_{ws} (resp., Σ_{wt}; we use the second letter, 't'). Define a *no-safe scenario* (resp., *no-stolen scenario*) as a scenario in which no credential's state is safe (resp., stolen). Denote the set of all no-safe (resp., no-stolen) scenarios by Σ_{ns} (resp., Σ_{nt}). The number of no-safe scenarios is $|\Sigma_{\mathsf{ns}}| = 3^n$ and all mechanisms fail in all of them (Lemma 3). Denote by $\overline{T}$ the complement set of T with respect to all scenarios. Note that $\overline{\Sigma_{\mathsf{ws}}} = \Sigma_{\mathsf{ns}}$ and $\overline{\Sigma_{\mathsf{wt}}} = \Sigma_{\mathsf{nt}}$.

Denote with-safe-with-stolen scenarios by $\Sigma_{\mathsf{wswst}} = \Sigma_{\mathsf{ws}} \cap \Sigma_{\mathsf{wt}}$; these play a special role in our proofs, so we count them now. By De-Morgan's law, $\overline{\Sigma_{\mathsf{wswst}}} = \Sigma_{\mathsf{ns}} \cup \Sigma_{\mathsf{nt}} = |\Sigma_{\mathsf{ns}}| + |\Sigma_{\mathsf{nt}}| - |\Sigma_{\mathsf{ns}} \cap \Sigma_{\mathsf{nt}}|$. Substitute $|\Sigma_{\mathsf{ns}}| = |\Sigma_{\mathsf{nt}}| = 3^n$ and $|\Sigma_{\mathsf{ns}} \cap \Sigma_{\mathsf{nt}}| = 2^n$ (as each credential is either leaked or lost). So we have

$$|\Sigma_{\mathsf{wswst}}| = |\Sigma| - |\overline{\Sigma_{\mathsf{wswst}}}| = 4^n - 2 \cdot 3^n + 2^n. \tag{2}$$

We can now prove Theorem 1.

Proof. If $n = 1$, there are 4 scenarios in total and all mechanisms fail in the 3 unsafe ones (Lemma 3). Hence $P(1) = 1$.

For $n \geq 2$, observe that for each with-safe-with-stolen scenario $\sigma \in \Sigma_{\mathsf{wswst}}$ (which exist for all $n \geq 2$) that a mechanism $M \in \mathcal{M}_n$ succeeds in, there is a complement $\overline{\sigma}$ where it fails (Lemma 2). Crucially, $\overline{\sigma} \in \Sigma_{\mathsf{wswst}}$ because a with-safe-with-stolen scenario has (at least) one safe and one stolen credential, and the safe, stolen states are switched in the complement. Therefore, M cannot succeed in more than half with-safe-with-stolen scenarios. The number of with-safe scenarios where all mechanisms fail is lower-bounded by $Q_{\mathsf{ws}}(n) = |\Sigma_{\mathsf{wswst}}|/2 \overset{\text{eq.(2)}}{=} \frac{4^n - 2 \cdot 3^n + 2^n}{2}$.

The number of scenarios in which all mechanisms fail is lower-bounded by $Q(n) = Q_{\mathsf{ws}}(n) + |\Sigma_{\mathsf{ns}}| = (4^n - 2 \cdot 3^n + 2^n)/2 + 3^n = (4^n + 2^n)/2$. Hence, the number of scenarios in which any mechanism succeeds is bounded by $P(n) = |\Sigma| - Q(n) = 4^n - \frac{4^n + 2^n}{2} = \frac{4^n - 2^n}{2}$. ◻

V. Maximal mechanisms

We now specify an approach to generate maximal mechanisms. These mechanisms wait for a bounded time allowing both players to submit credentials, therefore called *bounded-delay mechanisms*.

Any player can initiate authentication with a bounded-delay mechanism, which then starts a timer. Both U and A can send messages carrying credentials until the time elapses. Say the set of credentials submitted by player 0 is C_0 and player 1 is C_1. A deterministic *judging function* J selects the winner: It takes the two sets of credentials sent by the players and outputs an identifier, i.e., $J(C_0, C_1) \mapsto \{0, 1\}$. Bounded-delay mechanisms are realizable through the automaton model in a straightforward manner. Details are in App. B. Given a judging function J, we denote the corresponding mechanism by $M(J)$.

We present three properties of judging functions (§V-A) and show that they are sufficient for making the resultant mechanism maximally secure (§V-B). Then, we present three functions that produce, for any n, maximal mechanisms (§V-C).

A. Well-formed judging functions

Any bounded-delay mechanism with judging functions satisfying the following properties are maximal.

Definition 8 (ID-Agnostic (IA))**.** *A judging function J is* ID-Agnostic *if the identifier assignment does not affect the result, i.e., if $C \neq C'$ then $(J(C, C') = 0) \Leftrightarrow (J(C', C) = 1)$.*

If a judging function J satisfies IA, then we can compare any two sets of credentials and say that one of them is better than the other. We use the notation $C \succ_J C'$ (subscript omitted when obvious) to mean that C is *better* than C' according to J. Similarly, $C \succeq_J C'$ means that either the two credential sets are the same ($C = C'$) or C is better than C'.

Note that we do not say anything about the case where $C = C'$ in the above definition, so an ID-Agnostic function can choose to output either 0 or 1. WLOG all functions we present follow $\forall C : J(C, C) = 0$.

Definition 9 (Knowledge-rewarding (KR))**.** *A judging function is* knowledge rewarding *if, when the credentials submitted by one player are a strict subset of those submitted by the other, then it returns the latter, i.e., if $C' \subset C$ then $C' \prec C$.*

Definition 10 (Transitive knowledge-rewarding (TKR))**.** *A judging function is* transitive knowledge rewarding *if additional credentials cannot weaken a player's strategy, i.e., if $C_0 \neq C_1$, $C_1 \neq C_2$, $C_0 \succ C_1$, and $C_1 \succ C_2$, then $C_0 \not\subseteq C_2$.*

Note that $C_0 \neq C_2$ is implied above because if instead $C_0 = C_2$, then C_1 is both better and worse than C_0, which is not possible for an ID-Agnostic judging function.

A judging function with all the three properties and its resultant mechanism are *well formed*.

Definition 11 (Well-formedness)**.** *If a judging function J satisfies IA, KR, and TKR then it is* well-formed, *and the resulting bounded-delay mechanism $M(J)$ is* well-formed.

Note that a well-formed judging function J need not be *transitive*, i.e., it could allow: $C_0 \succ_J C_1$, $C_1 \succ_J C_2$ and $C_2 \succ_J C_0$. Examples of such functions appear later.

B. Maximality of well-formed mechanisms

We now prove that any well-formed mechanism is maximal. Two lemmas do a bulk of the work: Lemma 5 shows that the user can win in all with-safe-no-stolen scenarios and Lemma 8 shows that the user can win in exactly half of the with-safe-with-stolen scenarios. Using these results, Lemma 9 shows that the profile size of any n-credential well-formed mechanism is $P(n)$, which in turn implies maximality.

Before proving when well-formed algorithms succeed, we define submit-early strategies and prove Lemma 4, which we use throughout the rest of the proofs.

Definition 12 (Submit-early strategy)**.** *A submit-early strategy is a strategy where the player sends a single credential set when the automaton is in its initial state and nothing thereafter.*

The next lemma says that if one player employs a submit-early strategy with credentials C, then the only way the other player wins is by submitting a better or equal credential set.

Lemma 4. *Given a well-formed bounded-delay mechanism M and a scenario σ, if one player employs a submit-early strategy with credential set C, but the other player wins an execution, then the winning player must have submitted better or equal credentials C', i.e., $C' \succeq C$.*

Proof. Say player 0 submits credentials C as part of a submit-early strategy, but player 1 wins an execution. Denote the final set of credentials submitted by player 1 (perhaps across multiple messages) as C'.

Two cases emerge based on the order in which the two sets of credentials, C and C', are processed by the automaton. But, irrespective of the order, player 1 wins only if $C' \succeq C$: IA guarantees that, if on the contrary $C' \prec C$, then the automaton would not prefer player 1 irrespective of the player identifier (0 or 1) assigned to it. So player 1 cannot win an execution, and therefore, it must be that $C' \succeq C$. □

Denote by S^U_{all} (resp., S^A_{all}) the submit-early strategy where all credentials owned by the user C^U_σ (resp., attacker C^A_σ) are submitted. (S^A_{all} uses attacker-first ordering, as is the case whenever we do not explicitly specify it.)

We now prove the first major lemma.

Lemma 5. *Well-formed mechanisms succeed in with-safe-no-stolen scenarios, i.e., for all well-formed mechanisms M and scenarios $\sigma \in \Sigma_{\mathsf{wsnt}}$: $Suc(M, \sigma) = True$.*

Proof. Assume for contradiction that the mechanism fails in a with-safe-no-stolen scenario $\sigma \in \Sigma_{\mathsf{wsnt}}$. This means that for all user strategies the attacker wins at least one execution.

Say that the user follows the submit-early strategy S^U_{all} with all its credentials C^U_σ, then, due to Lemma 4, the attacker must know a set of credentials $C \subseteq C^A_\sigma$ such that $C \succeq C^U_\sigma$.

And since the scenario σ is a with-safe-no-stolen scenario, we have $C^A_\sigma \subset C^U_\sigma$, and since $C \subseteq C^A_\sigma$, we conclude that $C \subset C^U_\sigma$.

But due to the knowledge-rewarding (KR) property, $C \subset C^U_\sigma$ implies $C \prec C^U_\sigma$, thereby contradicting the previous statement $C \succeq C^U_\sigma$. Thus the attacker cannot win an execution in a with-safe-no-stolen scenario and the mechanism succeeds. □

Our next goal is to prove that the user wins in half of the with-safe-with-stolen scenarios Σ_{wswst}. Our approach is as follows. Recall that for every with-safe-with-stolen scenario σ, it's complement $\overline{\sigma}$ is also a with-safe-with-stolen scenario. Since any mechanism fails in one of σ or $\overline{\sigma}$ (Lemma 2), we use a winning attacker strategy in σ or $\overline{\sigma}$ to derive a winning user strategy in the other scenario.

First, we show that if there exists an attacker winning strategy for some execution, then the submit-early strategy with all credentials is also an attacker winning strategy for that execution.

Lemma 6. *Given a well-formed mechanism M, let the winning strategy of an attacker in a with-safe-with-stolen scenario $\sigma \in \Sigma_{\mathsf{wswst}}$ be $S^A \neq S^A_{\mathsf{all}}$, then S^A_{all} is also a winning strategy. That is, if $X = (M, \sigma)$ and $\forall S^U : Win_X(S^U, S^A) = A$ then $\forall S^U : Win_X(S^U, S^A_{\mathsf{all}}) = A$.*

Proof. Assume for contradiction the existence of a user strategy S^U such that the strategy S^A_{all} never wins an execution. By Lemma 4, it must be that the user submits a credential set $C^U \subseteq C^U_\sigma$ that is better than or equal to C^A_σ, i.e., $C^U \succeq C^A_\sigma$. Furthermore, the inequality is strict, i.e., $C^U \succ C^A_\sigma$, because if $C^U = C^A_\sigma$ then since $C^U \subseteq C^U_\sigma$, we get $C^A_\sigma \subseteq C^U_\sigma$, which is not true for a with-safe-with-stolen scenario $\sigma \in \Sigma_{\mathsf{wswst}}$.

Start again from the assumption. Since S^A is a successful attacker strategy, it must win an execution against the submit-early strategy with credential set C^U. And because of Lemma 4, the attacker must have submitted a credential set C^A such that $C^A \succeq C^U$.

We have two cases: $C^A = C^U$ or $C^A \succ C^U$. If $C^A = C^U$, since $C^U \succ C^A_\sigma$, we have $C^A \succ C^A_\sigma$. But KR says that if $C^A \subseteq C^A_\sigma$, then $C^A \preceq C^A_\sigma$, which leads to a contradiction.

Next, if $C^A \succ C^U$, since $C^U \succ C^A_\sigma$, the TKR property implies that $C^A \not\subseteq C^A_\sigma$ (note that $C^A \neq C^U$ and $C^U \neq C^A_\sigma$ hold, allowing the use of TKR.) But we have a contradiction as $C^A \subseteq C^A_\sigma$ by definition. So we conclude that the strategy S^A_{all} is also winning. □

Now, we prove that S^A_{all} is a winning strategy for the user in the complement scenario $\overline{\sigma}$.

Lemma 7. *If S^A_{all} is a winning strategy for the attacker in a with-safe-with-stolen scenario $\sigma \in \Sigma_{\mathsf{wswst}}$, then the messaging component of S^A_{all} is a winning strategy for the user in the complement with-safe-with-stolen scenario $\overline{\sigma}$.*

Proof. Since S^A_{all} is a winning strategy in scenario σ, it must win against any user strategy, including S^U_{all}. By Lemma 4, since the user follows a submit-early strategy, the attacker must have submitted a set of credentials C such that $C \succeq C^U_\sigma$. Since the attacker also follows a submit-early strategy S^A_{all}, the attacker submits credentials exactly once. Therefore, it must be that $C = C^A_\sigma$ and hence $C^A_\sigma \succeq C^U_\sigma$. The inequality is strict, i.e.,

$$C^A_\sigma \succ C^U_\sigma \ , \tag{3}$$

since $C^A_\sigma \neq C^U_\sigma$ for a with-safe-with-stolen scenario $\sigma \in \Sigma_{\mathsf{wswst}}$ and due to the mechanism's IA property.

Assume for contradiction that $\bar{S}^U_{\mathsf{all}} = S^A_{\mathsf{all}}$ (we mean that $\bar{S}^U_{\mathsf{all}}$ is the messaging component of S^A_{all}) is not a winning strategy for the user in the complement scenario $\overline{\sigma}$. It means that there exists an attacker strategy $\bar{S}^A$ that wins against $\bar{S}^U_{\mathsf{all}}$ in some executions of $\overline{\sigma}$. Applying Lemma 4 again, the attacker must have submitted a set of credentials $C \subseteq C^A_{\overline{\sigma}}$ such that $C \succeq C^U_{\overline{\sigma}}$. Since $\overline{\sigma}$ is also a with-safe-with-stolen scenario and due to the mechanism's IA property, the inequality is strict, i.e., $C \succ C^U_{\overline{\sigma}}$.

We claim that

$$C^A_{\overline{\sigma}} \succeq C^U_{\overline{\sigma}}. \tag{4}$$

This is because, if instead $C^U_{\overline{\sigma}} \succ C^A_{\overline{\sigma}}$, then because $C \succ C^U_{\overline{\sigma}}$ and TKR, we get $C \not\subseteq C^A_{\overline{\sigma}}$, which is false because $C \subseteq C^A_{\overline{\sigma}}$.

By definition, for any pair of complement scenarios, we have $C^U_{\sigma} = C^A_{\overline{\sigma}}$ and $C^U_{\overline{\sigma}} = C^A_{\sigma}$. Recasting eq. (4), we get $C^U_{\sigma} \succeq C^A_{\sigma}$, which contradicts eq. (3). Thus S^A_{all} is a winning user strategy in $\overline{\sigma}$. □

Now we can prove the second major lemma.

Lemma 8. *For all well-formed mechanisms M and with-safe-with-stolen scenarios $\sigma \in \Sigma_{\mathsf{wswst}}$, M succeeds in either σ or in its complement $\overline{\sigma}$, i.e., $Suc(M, \sigma) \vee Suc(M, \overline{\sigma})$.*

Proof. For each scenario $\sigma \in \Sigma_{\mathsf{wswst}}$, its complement $\overline{\sigma}$ satisfies $\overline{\sigma} \neq \sigma$ and $\overline{\sigma} \in \Sigma_{\mathsf{wswst}}$. Consider a pair of complement scenarios $\sigma, \overline{\sigma} \in \Sigma_{\mathsf{wswst}}$. No well-formed mechanism M succeeds in both σ and $\overline{\sigma}$ (Lemma 2). WLOG assume that M fails in σ, i.e., there exists a winning strategy S^A for the attacker that allows it to win an execution in σ. By Lemma 6, the existence of a winning strategy S^A implies that the submit-early strategy S^A_{all} is also a winning strategy. And by Lemma 7, the messaging component of S^A_{all} is a winning strategy for the user in the complement scenario $\overline{\sigma}$. □

We conclude by proving that the profile size of a well-formed mechanism is $P(n)$.

Lemma 9. *The profile size of any well-formed n-credential mechanism is $P(n)$.*

Proof. Lemma 5 shows that a well-formed mechanism succeeds in all with-safe-no-stolen scenarios ($\Sigma_{\mathsf{wsnt}} = \Sigma_{\mathsf{ws}} \cap \Sigma_{\mathsf{nt}}$). By basic set theory, $|\Sigma_{\mathsf{wsnt}}| = |\Sigma_{\mathsf{nt}}| - |\Sigma_{\mathsf{nt}} \setminus \Sigma_{\mathsf{ws}}|$ and $|\Sigma_{\mathsf{nt}} \setminus \Sigma_{\mathsf{ws}}| = |\Sigma_{\mathsf{nt}} \cap \overline{\Sigma_{\mathsf{ws}}}| = |\Sigma_{\mathsf{nt}} \cap \Sigma_{\mathsf{ns}}|$. Finally, since $|\Sigma_{\mathsf{nt}}| = 3^n$ and $|\Sigma_{\mathsf{nt}} \cap \Sigma_{\mathsf{ns}}| = 2^n$, we have $|\Sigma_{\mathsf{wsnt}}| = 3^n - 2^n$.

Lemma 8 shows that a well-formed mechanism succeeds in exactly half of the with-safe-with-stolen scenarios ($\Sigma_{\mathsf{wswst}} = \Sigma_{\mathsf{ws}} \cap \Sigma_{\mathsf{wt}}$). From eq. (2), $|\Sigma_{\mathsf{wswst}}| = (4^n - 2 \cdot 3^n + 2^n)$.

In total, a well-formed mechanism succeeds in $|\Sigma_{\mathsf{wsnt}}| + |\Sigma_{\mathsf{wswst}}|/2 = (4^n - 2^n)/2 = P(n)$ scenarios. □

Corollary 1. *Any well-formed mechanism is maximally secure.*

This is straightforward because the existence of a mechanism better than a well-formed mechanism (Lemma 9) violates the profile size bound (Theorem 1).

C. Algorithms for maximal mechanisms

We now specify three algorithms that produce maximal n-credential mechanisms for any n.

1) Priority mechanisms: Let a vector V define an ordering over all the credentials, then the priority judging function using V (Algorithm 2) decides on the player that submits a unique high-priority credential.

Algorithm 2 The priority judging function J^V_{pr}

```
Require: $V$ is a permutation over the elements of the set $\mathcal{C}_{\mathsf{all}}$.
  function $J^V_{\mathsf{pr}}(C_0, C_1)$                          ▷ $C_0 \subseteq \mathcal{C}_{\mathsf{all}}, C_1 \subseteq \mathcal{C}_{\mathsf{all}}$
      for $c = V_1, V_2, \ldots, V_n$ do
          if $c \in C_0 \wedge c \notin C_1$ then return 0        ▷ $C_0 \succ C_1$
          if $c \in C_1 \wedge c \notin C_0$ then return 1        ▷ $C_1 \succ C_0$
      return 0                                                    ▷ Default
```

Lemma 10. *Given a permutation V over elements of the set $\mathcal{C}_{\mathsf{all}}$, the priority judging function J^V_{pr} is well-formed.*

Proof. We prove each of the three properties (Definition 11). IA requires $(J^V_{\mathsf{pr}}(C_0, C_1) = 0) \Leftrightarrow (J^V_{\mathsf{pr}}(C_1, C_0) = 1)$. This holds because the priority function selects the unique high-priority credential irrespective of the order. KR requires $C_0 \subset C_1 \implies C_0 \prec C_1$. This holds because C_1 has at least one credential not in C_0 but C_0 has no credentials not in C_1.

To prove TKR, we need to show that given three distinct credential sets C_0, C_1 and C_2, if $C_0 \succ C_1$ and $C_1 \succ C_2$ then $C_0 \not\subseteq C_2$. By contradiction, assume $C_0 \subset C_2$ (since $C_0 \neq C_2$ by definition).

$C_0 \succ C_1$ implies the existence of a credential c that satisfies $c \in C_0 \setminus C_1$ such that $\{c\} \succ C_1 \setminus C_0$.

Observe that $C_0 \subset C_2$ implies $C_1 \setminus C_2 \subseteq C_1 \setminus C_0$.

We can rewrite the previous equation as $\{c\} \succ C_1 \setminus C_2$ due to the transitive nature of the priority judging function.

But the existence of $c \in C_0 \subset C_2$ such that $\{c\} \succ C_1 \setminus C_2$ implies that $C_2 \succ C_1$. This contradicts the TKR assumption $C_1 \succ C_2$, so $C_0 \subset C_2$ cannot be true and TKR is satisfied. □

Denote the priority mechanism corresponding to the judging function J^V_{pr} by M^V_{pr}. For example, the 2-credential priority mechanism $M^{[c_1,c_2]}_{\mathsf{pr}}$ along with its profile is in Fig. 1. Note that player 0 (resp., 1) can only take dashed (resp., dotted) edges and tries to reach f_0 (resp., f_1), and $\mathcal{R}$ denotes a clock reset. A proof for the profile computation is in App. C. Note that for any n, Algorithm 2 yields exactly one distinct mechanism because changes in the permutation V yield equivalent mechanisms, e.g., $M^{[c_1,c_2]}_{\mathsf{pr}} \cong M^{[c_2,c_1]}_{\mathsf{pr}}$.

2) Priority with exception: Mechanisms in this category are similar to priority mechanisms except when the last two credentials in the priority rule are submitted by the two players. For example, if the priority vector is $V = [c_2, c_3, c_1]$, then the exception is $\{c_1\} \succ \{c_3\}$. The resultant judging function is denoted by J^V_{pre} and is specified in Algorithm 3 (with the changes from Algorithm 2 highlighted). Well-formedness proof is in App. C. Like with priority mechanisms, this algorithm yields at most one maximal mechanism.

c_1 safe

c_2 \ c_3	St	Le	Lo	Sa
St	1	1	1	1
Le	1	1	1	1
Lo	1	1	1	1
Sa	1	1	1	1

c_1 lost

c_2 \ c_3	St	Le	Lo	Sa
St	0	0	0	0
Le	0	0	0	1
Lo	0	0	0	1
Sa	1	1	1	1

c_1 leaked

c_2 \ c_3	St	Le	Lo	Sa
St	0	0	0	0
Le	0	0	0	1
Lo	0	0	0	1
Sa	1	1	1	1

c_1 stolen

c_2 \ c_3	St	Le	Lo	Sa
St	0	0	0	0
Le	0	0	0	0
Lo	0	0	0	0
Sa	0	0	0	0

(a) $M_{\text{pr}}^{[c_1,c_2,c_3]}$

c_1 safe

c_2 \ c_3	St	Le	Lo	Sa
St	0	1	1	1
Le	1	1	1	1
Lo	1	1	1	1
Sa	1	1	1	1

c_1 lost

c_2 \ c_3	St	Le	Lo	Sa
St	0	0	0	0
Le	0	0	0	1
Lo	0	0	0	1
Sa	1	1	1	1

c_1 leaked

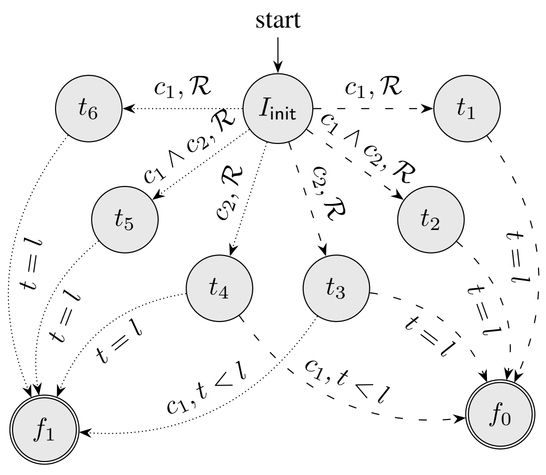
c_2 \ c_3	St	Le	Lo	Sa
St	0	0	0	0
Le	0	0	0	1
Lo	0	0	0	1
Sa	1	1	1	1

c_1 stolen

c_2 \ c_3	St	Le	Lo	Sa
St	0	0	0	0
Le	0	0	0	0
Lo	0	0	0	0
Sa	0	0	0	1

(b) $M_{\text{maj}}^{[c_1,c_2,c_3]}$

TABLE II: Profiles of a priority and a majority 3-credential mechanism



c_1 \ c_2	Stolen	Leaked	Lost	Safe
Stolen	0	0	0	0
Leaked	0	0	0	1
Lost	0	0	0	1
Safe	1	1	1	1

Fig. 1: $M_{\text{pr}}^{[c_1,c_2]}$ and its profile.

3) Majority mechanisms: Mechanisms in this category favor the player submitting the *most credentials*, so $C \succ C'$ if $|C| > |C'|$. To break ties, different strategies are possible which we model through a tie-breaking function $T(C_0, C_1) \mapsto \{0,1\}$. We only consider ID-Agnostic tie-breaking functions, i.e., if $0 \leftarrow T(C_0, C_1)$ then $1 \leftarrow T(C_1, C_0)$. The resultant judging function is denoted J_{maj}^{T} and is specified in Algorithm 4. The profile of a majority mechanisms $M_{\text{maj}}^{[c_1,c_2,c_3]}$, i.e., using priority vector $[c_1, c_2, c_3]$ to break ties, is in Tab. IIb. For comparison, we also show the profile of a priority mechanism $M_{\text{pr}}^{[c_1,c_2,c_3]}$ in Tab. IIa (they differ in just two scenarios). Well-formedness proof is in App. C.

Unlike the two priority-based mechanisms, this algorithm yields many distinct maximal mechanisms for all $n > 2$.

Algorithm 3 Priority with exception J_{pre}^{V}

```
Require: $V$ is a permutation over the elements of the set $\mathcal{C}_{\text{all}}$.
  function $J_{\text{pre}}^{V}(C_0, C_1)$          ▷ $C_0 \subseteq \mathcal{C}_{\text{all}}, C_1 \subseteq \mathcal{C}_{\text{all}}$
    if $C_0 = \{V_{n-1}\} \wedge C_1 = \{V_n\}$ then return 1
    if $C_0 = \{V_n\} \wedge C_1 = \{V_{n-1}\}$ then return 0
    for $c = V_1, V_2, \ldots, V_n$ do
      if $c \in C_0 \wedge c \notin C_1$ then return 0          ▷ $C_0 \succ C_1$
      if $c \in C_1 \wedge c \notin C_0$ then return 1          ▷ $C_1 \succ C_0$
    return 0          ▷ Default
```

The total number of majority mechanisms including credential permutations is $q(n) = 2^{\left(\binom{2n-1}{n-1} - 2^{n-1}\right)}$ (App. C). For $n = 3$, we manually discard equivalent mechanisms to find a total of 12 distinct majority mechanisms (out of $q(3) = 64$).

Algorithm 4 The majority judging function J_{maj}^{T}

```
  function $J_{\text{maj}}^{T}(C_0, C_1)$          ▷ $C_0 \subseteq \mathcal{C}_{\text{all}}, C_1 \subseteq \mathcal{C}_{\text{all}}$
    if $|C_0| > |C_1|$ then return 0
    if $|C_1| > |C_0|$ then return 1
    return $T(C_0, C_1)$          ▷ If $|C_0| = |C_1|$
```

VI. Complete Maximal Sets

A *complete maximal mechanism set* is a minimal set of n-credential mechanisms such that *any other n-credential mechanism is either equivalent to or worse than some mechanism in this set.*

Definition 13 (Complete set)**.** *A* complete maximal mechanism set *of n-credential mechanisms $\mathcal{O} = \{M_1, M_2 \ldots\} \subset \mathcal{M}_n$ satisfies three properties: (1) each $M \in \mathcal{O}$ is maximal; (2) any two distinct members are incomparable, i.e., $\forall M, M' \in \mathcal{O}, M \neq M' : M \nsim M'$; and (3) any n-credential mechanism is worse or equivalent to some mechanism in $\mathcal{O}$, i.e., $\forall M \in \mathcal{M}_n : \exists M' \in \mathcal{O}$ s.t. $M \preceq M'$.*

Several different complete maximal sets exist, but all their sizes are the same as each mechanism in a maximal set will have an equivalent in the other. The proof is in App. C.

We now present complete maximal sets for all $n \leq 3$. These sets are composed of bounded-delay mechanisms. For one-credential mechanisms, the attacker wins in three of the four possible scenarios (Lemma 3). Hence, the priority mechanism $M_{\mathsf{pr}}^{[c_1]}$ is the only one in the complete maximal set $\mathcal{O}_1$.

A. Two-credential mechanisms

The complete maximal set of 2-credential mechanisms is of size one. We find a set containing a priority mechanism.

Theorem 2. *A complete maximal set of 2-credential mechanisms is $\mathcal{O}_2 = \{M_{\mathsf{pr}}^{[c_1,c_2]}\}$.*

Proof. We prove $\mathcal{O}_2$ satisfies the three requirements (Definition 13). The first is that $M_{\mathsf{pr}}^{[c_1,c_2]}$ is maximal, which follows from Lemma 10. The second is that every pair of mechanisms in $\mathcal{O}_2$ must be incomparable to each other, which is vacuously true.

The third is that any 2-credential mechanism M must satisfy $M \preceq M_{\mathsf{pr}}^{[c_1,c_2]}$. Recall that $M \preceq M_{\mathsf{pr}}^{[c_1,c_2]}$ implies the existence of a mechanism $M' \cong M_{\mathsf{pr}}^{[c_1,c_2]}$ such that $\mathit{prof}(M) \subseteq \mathit{prof}(M')$. There are only two choices for M' that yield distinct profiles, namely $M' \in \{M_{\mathsf{pr}}^{[c_1,c_2]}, M_{\mathsf{pr}}^{[c_2,c_1]}\}$. And we are trying to prove that for any mechanism M, one of $\mathit{prof}(M) \subseteq \mathit{prof}(M_{\mathsf{pr}}^{[c_1,c_2]})$ or $\mathit{prof}(M) \subseteq \mathit{prof}(M_{\mathsf{pr}}^{[c_2,c_1]})$ is true.

We prove by contradiction. Assume there exists a mechanism M such that $\mathit{prof}(M) \not\subseteq \mathit{prof}(M_{\mathsf{pr}}^{[c_1,c_2]})$ and $\mathit{prof}(M) \not\subseteq \mathit{prof}(M_{\mathsf{pr}}^{[c_2,c_1]})$.

Consider the set of seven with-safe scenarios $\Sigma_{\mathsf{ws}} = \{\hat{\sigma}_1, \ldots, \hat{\sigma}_7\}$. Without loss of generality, let $\hat{\sigma}_1 = (\mathsf{safe}, \mathsf{stolen})$ (i.e., c_1 safe, c_2 stolen) and $\hat{\sigma}_7 = (\mathsf{stolen}, \mathsf{safe})$ is the complement of $\hat{\sigma}_1$. Figure 1 shows that $\mathit{prof}(M_{\mathsf{pr}}^{[c_1,c_2]}) = \{\hat{\sigma}_1, \ldots, \hat{\sigma}_6\}$ and $\mathit{prof}(M_{\mathsf{pr}}^{[c_2,c_1]}) = \{\hat{\sigma}_2, \ldots, \hat{\sigma}_7\}$.

By Lemma 3, the profile of M is a subset of Σ_{ws}, i.e., $\mathit{prof}(M) \subseteq \Sigma_{\mathsf{ws}}$. By basic set theory, the only way to guarantee $\mathit{prof}(M) \not\subseteq \mathit{prof}(M_{\mathsf{pr}}^{[c_1,c_2]})$ and $\mathit{prof}(M) \not\subseteq \mathit{prof}(M_{\mathsf{pr}}^{[c_2,c_1]})$ is if $\{\hat{\sigma}_1, \hat{\sigma}_7\} \in \mathit{prof}(M)$. But Lemma 2 rules this out as they are complement scenarios. □

Note 2. *The above mechanism can also be viewed as a majority mechanism with a priority rule to break ties or a priority with exception mechanism.*

B. Three-credential mechanisms

The complete maximal set of 3-credential mechanisms is of size 14. We group the constituents into majority and priority mechanisms, denoted by $\mathcal{O}_{\mathsf{maj},3}$ and $\mathcal{O}_{\mathsf{pr},3}$ respectively, i.e., $\mathcal{O}_3 = \mathcal{O}_{\mathsf{maj},3} \cup \mathcal{O}_{\mathsf{pr},3}$.

The set $\mathcal{O}_{\mathsf{pr},3}$ contains two priority-based mechanisms: the regular one (Algorithm 2) and with an exception (Algorithm 3).

The set $\mathcal{O}_{\mathsf{maj},3}$ contains 12 majority mechanisms that differ in their tie-breaking rule. Two tie rules are possible: linear priority, e.g., $c_1 \succ c_2, c_2 \succ c_3, c_1 \succ c_3$, or cyclic priority, e.g., $c_1 \succ c_2, c_2 \succ c_3, c_3 \succ c_1$ (last one switched). Note that a cyclic rule makes the resultant judging function non-transitive. The twelve majority mechanisms differ in the choice of tie rule used to break ties between 1-credential and 2-credential sets, e.g., one of them uses a linear rule for 1-credential sets and a cyclic rule for 2-credential sets. More details are in App. D.

Theorem 3. *A complete maximal set of 3-credential mechanisms is $\mathcal{O}_3 = \mathcal{O}_{\mathsf{pr},3} \cup \mathcal{O}_{\mathsf{maj},3}$.*

Proving the first two requirements of Definition 13 is straightforward. We prove the last requirement that no mechanism is better than a mechanism in $\mathcal{O}_3$ by contradiction. Assume that a mechanism M exists such that it is incomparable with any mechanism in $\mathcal{O}_3$. We encode this as a constraint in a constraint solver and have it search for a satisfying profile. We also add another constraint relying on the observation that if M fails in a scenario σ, then it must fail in all scenarios $\widehat{\sigma}$ that are worse than σ where $\widehat{\sigma}$ is *worse* than σ if the attacker knows the same or more credentials whereas the user knows the same or fewer credentials. We find that the constraint solver is unable to find a solution, therefore no such mechanism exists. The details are in App. D.

Note 3. *It remains an open question how to analytically find complete maximal sets for larger number of credentials.*

VII. Applications

We now apply our framework to analyze a popular cryptocurrency wallet (§VII-A), a bank account (§VII-B) and all known 2-credential mechanisms (§VII-C).

A. The Argent Cryptocurrency Wallet

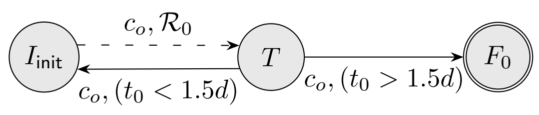
Social recovery is a prominent approach [32] to design non-custodial cryptocurrency wallets where a user's social circle, e.g., friends and family, is used to manage keys. We analyze Argent [9], a popular social-recovery wallet [19].

We describe Argent's operation as an automaton (§VII-A1), find the resultant profile (§VII-A2), and discuss security improvements (§VII-A3).

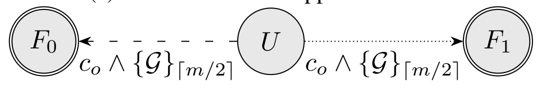
1) Operation: The Argent mechanism M_{Arg}^{m} uses $m+1$ credentials, consisting of an owner credential c_o and a set of m so-called *guardian* credentials $\mathcal{G} = \{g_1, g_2, \ldots, g_m\}$. The owner credential is a cryptographic key on the user's mobile phone. Guardians can be anything from a friend's cryptocurrency wallet to a hardware wallet or a paid third-party service. Thus, in our notation, $\mathcal{C}_{\mathsf{all}} = \{c_o, g_1, g_2, \ldots, g_m\}$.

The user maintains a list of *approved addresses* (called *trusted contacts*) to which she can send funds immediately. Transferring funds to a unapproved address requires additional steps. The Argent wallet allows adding or removing approved addresses, adding or removing guardians, replacing the owner credential (if her phone is lost or stolen) and transferring deposited funds.

We review all the attacker's options to withdraw funds into an unapproved address. (withdrawing to approved addresses raises the question of how those are protected, which is not

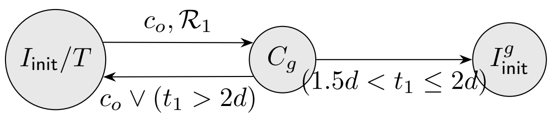


(a) Transfer to an unapproved address

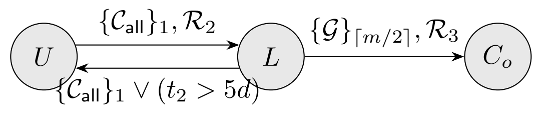


F_0
U
F_1
$c_o \wedge \{\mathcal{G}\}_{\lceil m/2 \rceil}$
$c_o \wedge \{\mathcal{G}\}_{\lceil m/2 \rceil}$

(b) Immediately transfer to an unapproved address

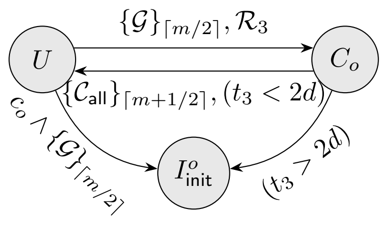


(c) Change guardians



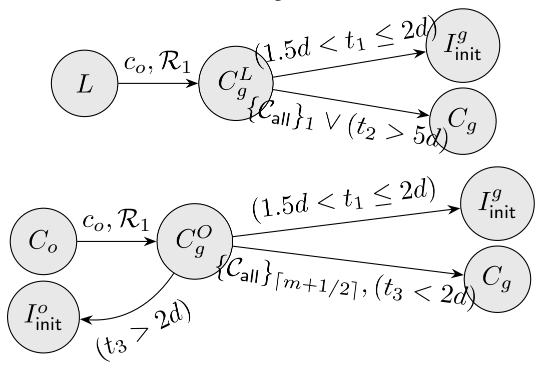
U
$\{\mathcal{C}_{\mathsf{all}}\}_1, \mathcal{R}_2$
L
$\{\mathcal{G}\}_{\lceil m/2 \rceil}, \mathcal{R}_3$
C_o
$\{\mathcal{C}_{\mathsf{all}}\}_1 \vee (t_2 > 5d)$

(d) Locking wallet



$\{\mathcal{G}\}_{\lceil m/2 \rceil}, \mathcal{R}_3$
U
C_o
$\{\mathcal{C}_{\mathsf{all}}\}_{\lceil m+1/2 \rceil}, (t_3 < 2d)$
$c_o \wedge \{\mathcal{G}\}_{\lceil m/2 \rceil}$
I^o_{init}
$(t_3 > 2d)$

(e) Change owner



(f) Delete guardians when locked

Fig. 2: Argent mechanism M^m_{Arg} sub-automata

well defined.) To simplify the presentation, we first review different paths separately. Figure 2 depicts five automata, each capturing a different functionality of the Argent mechanism.

Dashed (resp., dotted) edges indicates an edge that can be taken only by player 0 (resp., 1). Solid edges can be taken by either player. The state I_{init} is the starting state of the automaton. Two final states exist: F_0 (resp., F_1) stands for the state in which player zero (resp., player one) wins. We only show half of each automaton, the portion assuming the first move is made by player 0; we omit the other half because it is symmetric. Reaching a final state signifies a successful withdrawal. Note that we use multiple clocks to simplify the presentation: in particular, $\mathcal{R}_i$ denotes a reset of ith clock, i.e., $t_i \leftarrow 0$. (Using multiple clocks does not impact our formalization as they can be removed to construct a standard automaton [31].)

If credentials are unchanged, there are only two ways for the attacker to withdraw funds to an address it controls (Fig. 2a, Fig. 2b). She can add this address to the approved list and then withdraw, a process we call *slow withdrawal*. In order to do so, the owner needs to initiate an action (I_{init} to T). The addition becomes effective after 1.5 days, at which point the owner can withdraw (T to F_0). The approval process can be cancelled with the owner credential during the 1.5 day period (T to I_{init}): this is useful if the owner credential c_o is leaked.

Another way to withdraw funds is to order a *fast withdrawal* with the owner credential and at least half of the guardian credentials (Fig. 2b).

The attacker can also try to withdraw funds by first changing the set of credentials. One way is to change the set of guardians (Fig. 2c). The owner initiates the process (I_{init} to C_g) and can be finalized only during a time window that starts 1.5 days after initiation and lasts for 12 hours thereafter. As in the slow withdrawal case, the guardian change process can be cancelled by c_o.

Note that a guardian change implies a change in the underlying scenario. The execution moves to another copy of the same automaton, indicated by the new start state I^g_{init}, with a new scenario σ' that reflects the change to new guardians. For example, if player 0 is able to add a guardian, then the new scenario σ' has a new guardian credential g_{m+1} that is known to 0 alone, and $\mathcal{C}'_{\mathsf{all}} = \mathcal{C}_{\mathsf{all}} \cup \{g_{m+1}\}$.

Argent implements *locking*, a feature intended for situations when the owner suspects a credential fault [9]. As shown in Fig. 2d, the mechanism can be moved from any state into a locked state with any one credential. A limited set of actions are possible in the locked state, namely, change owner and unlock. Unlocking the automaton, i.e., moving back into its previous state can be done with any credential. The state U represents all unlocked states, i.e., I_{init} or T or C_g.

Argent allows changing the owner, a process they call *recovery*. There are two ways to do so (Fig. 2e). If the current owner credential is lost, the automaton can be moved from any unlocked state to the recovery state C_o with $\lceil m/2 \rceil$ guardian credentials. The new owner address is specified in this step. Finalizing this new owner takes 2 days. In this period, owner change can be cancelled with $\lceil (m+1)/2 \rceil$ credentials, which can include the original owner: this is useful if the owner credential was not actually lost.

The second way is for when the owner credential is safe, e.g., if the user wants to transfer the Argent app between phones. In this case, there is no need to wait. The owner can be changed immediately with the owner credential c_o and $\lceil m/2 \rceil$ guardians, a process we call *fast owner-change*.

Similar to when a guardian change happens, a change of owner moves the execution back to a start state I^o_{init} with a modification to the scenario. For example, if player 0 is able to change the owner, then the new scenario will reflect that: the new credential c_o is known to player 0 alone.

Finally, Argent allows guardian revocation (but not addition) from a locked (or recovery) state (Fig. 2f). Confirming revocation can be done after 1.5 days like before, but canceling it requires unlocking (or canceling recovery) first.

2) Profile analysis: We now analyze the security of Argent M^m_{Arg} with one and with two guardians ($m = 1, 2$).

The one guardian case has just two credentials: an owner c_o and a guardian g_1. The user can reach a final state in two ways: (a) submit both the credentials (Fig. 2b) or (b) submit just c_o and wait for 1.5 days (Fig. 2a). So if one of the credentials is lost but the other is safe, then the user's winning strategy is to revoke the lost credential. This is winning because all the credentials will be safe in the new automaton.

If c_o is safe and g_1 is leaked or stolen, the mechanism succeeds and the user's winning strategy is to initiate guardian revocation from any state. In particular, even if the attacker initiates an owner change before (I_{init} to C_o), the user can initiate guardian revocation (reach C^O_g). The user does not even need to cancel the owner change as it requires waiting 2 days whereas the unsafe guardian can be revoked after 1.5 days.

If g_1 is safe and c_o is leaked or stolen, the mechanism fails and the attacker's winning strategy is to lock the automaton if the current state is unlocked, and cancel a recovery if the current state is C_o (i.e., owner change was initiated).

In summary, the profile of 1-guardian Argent M^1_{Arg} has 5 scenarios, and M^1_{Arg} is worse than our maximal mechanism with 6 scenarios, i.e., $M^1_{\mathsf{Arg}} \prec M^{[c_o,g_1]}_{\mathsf{pr}}$ (Fig. 1). We similarly analyze 2-guardian Argent M^2_{Arg} in App. E to find that its profile has 22 scenarios, and that it is worse than a priority mechanism with 28 scenarios, i.e., $M^2_{\mathsf{Arg}} \prec M^{[c_o,g_1,g_2]}_{\mathsf{pr}}$ (Tab. IIa).

3) Improving Argent: We propose a simple strategy to improve Argent's profile: executing multiple transactions atomically, commonly known as a multicall [33]. For example, consider the scenario (c_o leaked, g_1 safe) where Argent previously failed. It succeeds now because the user can atomically execute two transitions: unlock and fast withdrawal from a locked state (L or C_o). With a multicall, Argent becomes maximal with 1 guardian, but not with 2 guardians or more. For example, with two guardians, the profile of Argent with multicalls has 24 scenarios, weaker than 28 in a maximal mechanism.

While it is technically feasible to run a multicall with Argent's contracts today (e.g., using Uniswap's multicall contract [33]), we could not find a mention of this technique in Argent's documentation and Argent does not natively support it.

As noted before, the use of multicalls only helps improve Argent's profile to an extent. One can attain better security with our maximal mechanisms to achieve the same functionality as Argent, except we omit locking as it does not improve security in our model. Rather than using different mechanisms for the various functions, e.g., change owner / guardian, we propose the use of a maximal mechanism for all functionalities, thus vastly simplifying the design. We propose the use of a majority mechanism with some tie-breaking function due to its simplicity.

One missing feature from our proposed design is fast withdrawals (Fig. 2b) or fast owner change (Fig. 2e). However, it is easy to add it: simply move to the final state if *all the credentials* are submitted. The modified mechanism is still maximal, i.e., doesn't break PA, KR and TKR, since the fast path can only be enabled if all the credentials are provided. However, it does incur a usability hit: for all $m > 1$, the user needs to do more work in gathering guardian approvals than with Argent. (If $m = 1$, the user needs to submit all the credentials anyway.)

We demonstrate the practicality of our mechanisms with a proof-of-concept priority mechanism in Solidity (App. A). Our implementation takes about 100 lines of code and requires 210k gas for the costliest function, compared to 150k for Argent ($7.6 and $5.4 resp., assuming a gas price of 30gwei and Ethereum price of $1200).

B. HDFC Online Bank Account

We model the authentication mechanisms used by the HDFC bank website [10]. The bank provides a web portal for the users to login with their password credential c_p. We assume that 2FA is enabled, in particular, one time PINs are received on the registered mobile number; this credential is denoted c_m. The bank allows users to add additional factors if needed, e.g., email, but we do not model it for simplicity. Finally, users can change the registered mobile number by submitting a request at the bank along with an ID proof [34]. We model any acceptable identity proof using the credential c_{id}(modeled in the first automaton in Fig. 4.) There are other ways to change the mobile number [35], e.g., use a debit card and visit an ATM, but again, we do not model it for simplicity.

Like before, we focus on how to withdraw money to a new (attacker-controlled) account number. It involves using the password c_p to login and authenticating via mobile c_m to initiate the third-party addition process. This process takes 30 minutes, during which it can be cancelled by logging in to the bank portal. After the time elapses, money can be transferred to the new account. This requires another 2FA.

The bank allows changing the password after authenticating via mobile. In order to change the registered mobile number, the user must submit a form along with relevant identity proof c_{id}, and the change happens after 3 days [35].

We now compute the profile assuming c_{id} is safe. If c_m is safe, then the mechanism succeeds irrespective of the state of c_p. The user's winning strategy is to change the password (notice that the attacker cannot initiate any action with c_p alone). The mechanism also succeeds if c_m is lost because c_{id} is safe, and it can be used to change the mobile number after three days. In other scenarios, the mechanism fails because the three day delay is sufficient to withdraw money by the attacker.

We find that the mechanism's success does not change irrespective of the state of the password credential c_p, hence

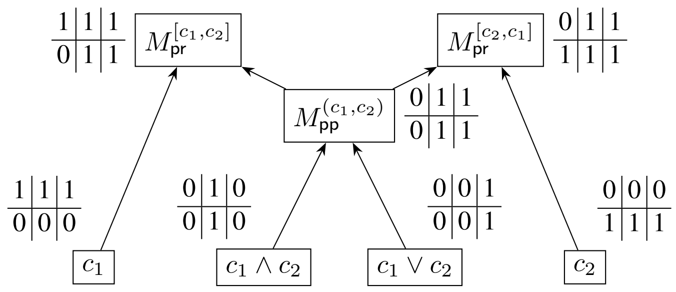


Fig. 3: One and two credential mechanisms.

it can be safely removed. While the redundancy of passwords was observed by prior work [36], ours is the first to prove it in a rigorous manner to our knowledge. In summary, HDFC's profile has 20 scenarios, weaker than our maximal mechanism. The automaton, profile and a complete analysis is in App. E.

C. Prior Two-Credential Mechanisms

We compare (Fig. 3) all distinct n-credential mechanisms where $n \leq 2$ (that we know of) in a single graph, covering common approaches used in practice, mechanisms from recent prior work, and our new ones. Each rectangle represents a mechanism. Arrows between rectangles show the $\succ$ relation, i.e., an arrow from mechanism M_1 to M_2 signifies $M_2 \succ M_1$. Adjoining a node is the profile of the mechanism restricted to with-safe scenarios. The first row corresponds to c_1 being safe and c_2 being stolen, leaked or lost in that order, and the reverse for the second row.

The bottom row in the figure shows standard approaches to storing a private key: either you store it as is, or split the key ($c_1 \wedge c_2$), or keep two separate copies ($c_1 \vee c_2$).

The middle row represents a Paralysis Proofs [14], which is the most secure 2-credential mechanism from prior work to the best of our knowledge. We model their mechanism, denoted by $M_{\mathsf{pp}}^{(c_1,c_2)}$, in App. E. With two credentials, they use an AND-mechanism $c_1 \wedge c_2$ with an additional feature. Any credential can be challenged, and if no response is received within a fixed duration, then the challenged credential is removed. For example, if c_1 is successfully challenged, the new mechanism becomes just c_2. In this manner, they handle credential loss, and achieve better security than both AND and OR mechanisms. But they do not encode the priority between credentials, so it fails in both the scenarios where one credential is safe and another is stolen.

Finally, the top row contains our two maximal mechanisms, namely, $M_{\mathsf{pr}}^{[c_1,c_2]}$ and its isomorphism. As Fig. 3 illustrates, our 2-credential mechanisms achieve better security.

VIII. Conclusion and Practical Implications

We formalize the authentication problem in a synchronous environment and define the security profile for evaluating authentication mechanisms. After bounding the profile size for any number of credentials n, we discover three types of mechanisms that achieve this bound and show they cover all maximal mechanisms for $n \leq 3$. Our analysis allows improving existing interactive authentication protocols and we provide an Ethereum implementation.

An immediate insight arising from this work is the criticality of the assumed synchronous channel. In practice, such a channel must be implemented. For example, consider email serving as the notification channel from a bank to its clients. If the attacker gained even temporary access to a user's mobile device, she could delete any notifications, thus voiding the channel's effectiveness. We propose to overcome this vulnerability by making such notification emails *sticky* – they cannot be deleted for, say, 24 hours. In other settings such as blockchain smart contracts, the notifications are public, but the user should use multiple devices to monitor the chain [37] (as used for other goals, e.g., [38], [39], [40]).

Our results open the door to a host of questions on authentication mechanisms in alternative models, e.g., with asynchronous communication, partial knowledge [27], and taking into account economic incentives [26]. Exploring these questions and harnessing our results to strengthen existing systems will bolster the security of digital assets and online services.

References

[1] M. Bishop, *Introduction to Computer Security*. Addison-Wesley, 2004.
[2] F. B. Schneider, "Authentication for people (draft textbook chapter)," https://www.cs.cornell.edu/fbs/publications/chptr.AuthPeople.pdf, 2009, retrieved Oct'22.
[3] J. Bonneau, C. Herley, P. C. Van Oorschot, and F. Stajano, "The quest to replace passwords: A framework for comparative evaluation of web authentication schemes," in *2012 IEEE S&P*, 2012, pp. 553–567.
[4] E. V. Mangipudi, U. Desai, M. Minaei, M. Mondal, and A. Kate, "Uncovering impact of mental models towards adoption of multi-device crypto-wallets," *Cryptology ePrint Archive*, 2022.
[5] J. Colnago, S. Devlin, M. Oates, C. Swoopes, L. Bauer, L. Cranor, and N. Christin, ""it's not actually that horrible": Exploring adoption of two-factor authentication at a university," in *In ACM CHI*, 2018, p. 1–11.
[6] Federal Trade Commission, "Consumer sentinel network data book 2021," February 2022.
[7] Chainalysis, "60% of Bitcoin is held long term as digital gold. what about the rest?" https://blog.chainalysis.com/reports/bitcoin-market-data-exchanges-trading/, June 2020.
[8] P. Jha, "The aftermath of Axie Infinity's $650m Ronin bridge hack," *Cointelegraph*, 2022, https://cointelegraph.com/news/the-aftermath-of-axie-infinity-s-650m-ronin-bridge-hack.
[9] "Argent specification," https://github.com/argentlabs/argent-contracts/blob/develop/specifications/specifications.pdf, April 2021.
[10] "Cooling period: Get time to review newly added beneficiaries," https://www.hdfcbank.com/personal/useful-links/security/security-measures/cooling-period, 2022.
[11] Internal Revenue Service, "Taxpayer guide to identity theft," https://www.irs.gov/newsroom/taxpayer-guide-to-identity-theft, 2022.
[12] I. Eyal, "On cryptocurrency wallet design," in *Tokenomics*, 2021, pp. 4:1–4:16.
[13] S. Hammann, S. Radomirović, R. Sasse, and D. Basin, "User account access graphs," in *ACM CCS*, 2019, pp. 1405–1422.
[14] F. Zhang, P. Daian, I. Bentov, I. Miers, and A. Juels, "Paralysis proofs: Secure dynamic access structures for cryptocurrency custody and more," in *AFT*, 2019, pp. 1–15.
[15] L. O'Gorman, "Comparing passwords, tokens, and biometrics for user authentication," *Proceedings of the IEEE*, vol. 91, no. 12, pp. 2021–2040, 2003.
[16] C. Harper, "Multisignature wallets can keep your coins safer (if you use them right)," 2020, Coindesk.
[17] R. Gennaro, S. Goldfeder, and A. Narayanan, "Threshold-optimal DSA/ECDSA signatures and an application to Bitcoin wallet security," in *ACNS*, 2016, pp. 156–174.

[18] T. Be'ery, "Threshold signatures: The future of private keys," https://zengo.com/threshold-signatures-the-future-of-private-keys/, 2018.

[19] E. Banulescu, "Argent wallet: Everything you need to know," July 2022, https://beincrypto.com/learn/argent-wallet/.

[20] C. Jacomme and S. Kremer, "An extensive formal analysis of multi-factor authentication protocols," *ACM TOPS*, vol. 24, no. 2, pp. 1–34, 2021.

[21] M. Barbosa, A. Boldyreva, S. Chen, and B. Warinschi, "Provable security analysis of FIDO2," in *CRYPTO*, 2021, pp. 125–156.

[22] J. Bonneau, A. Miller, J. Clark, A. Narayanan, J. A. Kroll, and E. W. Felten, "SoK: Research perspectives and challenges for Bitcoin and cryptocurrencies," in *IEEE S&P*, 2015, pp. 104–121.

[23] J.-P. Aumasson, A. Hamelink, and O. Shlomovits, "A survey of ECDSA threshold signing," *Cryptology ePrint Archive*, 2020.

[24] S. Appelcline, "Using timelocks to protect digital assets," https://github.com/BlockchainCommons/SmartCustody/blob/master/Docs/Timelocks.md, 2021, [Accessed Sep 2022].

[25] P. Baratam, "Secure cryptocurrency exchange & wallet," https://www.coinvault.tech/wp-content/uploads/2020/10/CoinVault-Secure-Cryptocurrency-Exchange.pdf, 2020.

[26] M. Möser, I. Eyal, and E. Gün Sirer, "Bitcoin covenants," in *FC*, 2016.

[27] S. Blackshear, K. Chalkias, P. Chatzigiannis, R. Faizullabhoy, I. Khaburzaniya, E. K. Kogias, J. Lind, D. Wong, and T. Zakian, "Reactive key-loss protection in blockchains," in *WTSC@FC*, 2021, pp. 431–450.

[28] M. Blaze, J. Feigenbaum, and J. Lacy, "Decentralized trust management," in *IEEE S&P*, 1996, pp. 164–173.

[29] R. L. Rivest and B. Lampson, "SDSI - A simple distributed security infrastructure," in *USENIX Security*, 1996.

[30] M. Blaze, J. Feigenbaum, and A. D. Keromytis, "KeyNote: Trust management for public-key infrastructures," in *Security Protocols*, 1998, pp. 59–63.

[31] R. Alur and D. L. Dill, "A theory of timed automata," *Theor. Comput. Sci.*, vol. 126, no. 2, pp. 183–235, 1994. [Online]. Available: https://doi.org/10.1016/0304-3975(94)90010-8

[32] V. Buterin, "Why we need wide adoption of social recovery wallets," https://vitalik.ca/general/2021/01/11/recovery.html, 2021.

[33] "Multicall | Uniswap." [Online]. Available: https://docs.uniswap.org/protocol/reference/periphery/base/Multicall

[34] "How to update contact details at a branch," 2022, https://www.hdfcbank.com/personal/useful-information/change-contact-details.

[35] "How to change mobile number in HDFC bank: 2 easy ways," 2022, https://thebankhelp.com/how-to-change-mobile-number-in-hdfc-bank/.

[36] S. Pearman, S. A. Zhang, L. Bauer, N. Christin, and L. F. Cranor, "Why people (don't) use password managers effectively," in *SOUPS*, 2019, pp. 319–338.

[37] "How to get notified on ethereum," https://vittominacori.medium.com/how-to-get-notified-on-ethereum-or-tokens-received-4b71859a064b.

[38] P. McCorry, S. Bakshi, I. Bentov, S. Meiklejohn, and A. Miller, "Pisa: Arbitration outsourcing for state channels," in *AFT*, 2019, pp. 16–30.

[39] M. Khabbazian, T. Nadahalli, and R. Wattenhofer, "Outpost: A responsive lightweight watchtower," in *AFT*, 2019, pp. 31–40.

[40] L. Gudgeon, P. Moreno-Sanchez, S. Roos, P. McCorry, and A. Gervais, "SoK: Layer-two blockchain protocols," in *FC*, 2020, pp. 201–226.

Appendix A
Priority Mechanism Smart Contract

A proof-of-concept implementation of the priority mechanism is below. Note that this is strictly an academic prototype meant to elucidate its inner workings, not to be used in production environments.

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.7.0 <0.9.0;

contract PriorityWallet {
    uint public numCredentials;
    mapping(address => uint) public priority;
    bool public withdrawalInProgress = false;
    uint public maxClaimID = 0;
    struct Details {
        bool[] supporters;
        uint amount;
        address payable addr;
    }
    Details[] public claimDetails;
    uint64 constant public delay = 30;
    uint64 public expiryTime;

    // Set the guardians and the priority vector. Ideally, you'd also want to deposit some money.
    constructor(address[] memory credentialList_) payable {
        numCredentials = credentialList_.length;
        for (uint i = 0; i < credentialList_.length; i++) {
            priority[credentialList_[i]] = i + 1;
        }
    }
    function getBalance() public view returns (uint) {
        return address(this).balance;
    }
    // start a new withdrawal. Internally creates a new claim
    function initiateWithdrawal() public {
        assert (priority[msg.sender] > 0);
        assert (!withdrawalInProgress);
        withdrawalInProgress = true;
        expiryTime = uint64(block.timestamp + delay);
        // purge previous claim data (if any)
        delete claimDetails;
        maxClaimID = 0;
    }
    function getClaimSupporters(uint claimID) public view returns (bool[] memory) {
        return claimDetails[claimID].supporters;
    }
    // adds approval to an existing claim
    function addApproval(uint claimID) public {
        assert (withdrawalInProgress);
        assert (claimID < maxClaimID);
        assert (uint64(block.timestamp) <= expiryTime);
        assert (priority[msg.sender] > 0);
        claimDetails[claimID].supporters[priority[msg.sender]] = true;
    }
    function createNewClaimForWithdrawal(uint amount, address payable ToAddress) public returns (uint claimID) {
        assert (priority[msg.sender] > 0);
        assert (withdrawalInProgress); // otherwise call initiateWithdrawal
        assert (uint64(block.timestamp) <= expiryTime);

        bool[] memory supporters = new bool[](numCredentials + 1);
        supporters[priority[msg.sender]] = true;
        claimDetails.push(Details(supporters, amount, ToAddress));

        claimID = maxClaimID; // create new claimID
        maxClaimID = maxClaimID + 1;
    }
    function withdraw() public {
        assert(uint64(block.timestamp) > expiryTime);
        assert(withdrawalInProgress);
        bool[] memory potentialWinners = new bool[](maxClaimID);
        for (uint c = 0; c < maxClaimID; c++) {
            potentialWinners[c] = true;
        }
        for (uint p = 1; p <= numCredentials; p++) {
            bool foundAny = false;
            for (uint c = 0; c < maxClaimID; c++) {
                if (potentialWinners[c] &&
                    claimDetails[c].supporters[p]) {
                    foundAny = true;
                }
            }
            if (foundAny) {
                for (uint c = 0; c < maxClaimID; c
                    ++) {
                    if (potentialWinners[c] && !
                        claimDetails[c].supporters[p
                        ]) {
                        potentialWinners[c] = false;
                    }
                }
            }
        }
        for (uint c = 0; c < maxClaimID; c++) {
            if (potentialWinners[c]) {
                uint winningClaimID = c;
                bool sent = claimDetails[
                    winningClaimID].addr.send(
                    claimDetails[winningClaimID].
                    amount);
                require(sent, "Failed to send Ether"
                    );
                withdrawalInProgress = false;
                break;
            }
        }
    }
}
```

Appendix B
Bounded-delay automaton construction

Given a judging function J, we now explain how to construct a bounded-delay mechanism automaton $M(J)$.

The high-level idea is as follows. The automaton has states in two levels below the start state. The intermediate level contains a state for each combination of player and set of credentials submitted, forming the children of the start state I_{init}. Then, for each intermediate state, we find all possible credential sets the second player can submit in order to win and add outgoing edges correspondingly. There are only two final states, one each for the respective players.

Consider the example automaton in Fig. 1. It corresponds to the following mechanism: Given two credentials c_1, c_2, consider the judging function J that prioritizes c_1 over c_2. As shown, the start state has six children corresponding to the 3 possible credential sets $(c_1, c_2, c_1 \wedge c_2)$ from the 2 parties (0, 1). These six states form the intermediate states. Only two final states f_0 and f_1 exist. Player 0 (resp., 1) wins if the execution reaches f_0 (resp., f_1). Player guards are depicted through dashed and dotted edges: 0 can only take dashed edges while 1 can only take dotted edges.

Note that we assume that each party can only submit one message. This is done to simplify the automaton construction.

We briefly explain the construction of a bounded-delay mechanism from a judging function. Given a judging function J defined over a set of credentials $\mathcal{C}_{\mathsf{all}}$, we describe an authentication mechanism $M = (\mathcal{C}_{\mathsf{all}}, \mathcal{T}, clock, \mathcal{D}, \mathcal{T}_0^{\mathsf{fin}}, \mathcal{T}_1^{\mathsf{fin}})$. Note that we assume that each player submits all their credentials at once to simplify the construction.

Let the set of all AND-credential-guards that use $\wedge$ connector only be G. $|G| = 2^n - 1$ as each credential can either be present or absent and we omit the ε-transition.

The set of all states $\mathcal{T}$ consists of $2 \cdot (2^n - 1)$ intermediate states and 2 final states. An intermediate state t is created for each player ID guard $g^{plr} \in \mathcal{P}$ and credential guard $g_\gamma^{cd} \in G$, with a transition between the start state and this new state, $(I_{\mathsf{init}}, g^{plr}, g_\gamma^{cd}, \bot, \mathsf{True}, t)$, i.e., no clock guard, with clock reset.

The two final states are f_0, f_1. The set $\mathcal{T}_0^{\mathsf{fin}}$ contains f_0 and the set $\mathcal{T}_1^{\mathsf{fin}}$ contains f_1.

The set of all transitions $\mathcal{D}$ consists of edges between the start and intermediate states (explained before) and those between the intermediate and final states (explained next).

There are two different types of transitions between the intermediate and final states. The first type allows the first-mover to win but only after some time elapses. For each intermediate state t, if the player identifier that submitted credentials before is $\mathsf{id}_\gamma \in \mathcal{P}$, then there is a transition $(t, p, \bot, t = l, \mathsf{False}, f_\gamma)$, i.e., no credential guard.

The second type allows the other party to win, but only if they submit better credentials. Let C_γ denote the set of credentials submitted by the player id_γ (in g_γ^{cd}) to reach an intermediate state t. Let $p' = \mathsf{id}_{1-\gamma}$.

Find the set of credential guards G' that result in the second-mover winning, i.e., if $\gamma = 0$, find all $g_1^{cd} \in \mathcal{G}_c$ such that C_1 contains the credentials in g_1^{cd} and $J(C_0, C_1) = 1$. And if $p = 1$, find g_1^{cd} s.t. $J(C_1, C_0) = 0$. Add all such guards to G'. For each $g^{cd} \in G'$, there is a transition $(t, p', g^{cd}, t < l, \mathsf{False}, f_{1-\gamma})$.

Appendix C
Other Results

Proofs of well-formedness are in §C-A, some miscellaneous results in §C-B, and a count of majority functions is in §C-C.

A. Well-formedness proofs

Lemma 11. *Given a permutation V over set $\mathcal{C}_{\mathsf{all}}$, the priority with exception judging function J_{pre}^V is well-formed.*

Proof. It is straightforward to see that the judging function satisfies IA and KR. To satisfy TKR, we want to show that given three different credential sets C_0, C_1 and C_2, if $C_0 \succ C_1$ and $C_1 \succ C_2$ then $C_0 \not\subseteq C_2$.

Since $C_0 \succ C_1$, we have two cases: (A) $C_0 \succ C_1$ is not the exception or (B) $C_0 \succ C_1$ is the exception. Similarly $C_1 \succ C_2$ means: (I) $C_1 \succ C_2$ is not the exception or (II) $C_1 \succ C_2$ is the exception. One of the four cases: A-I, B-I, A-II and B-II must be true. We consider each one separately.

The proof for the case A-I (no exceptions) is exactly the same as the one in Lemma 10.

Case B-II (both exceptions) is impossible because there is only one exception and the three credential sets are different.

The case B-I where $C_0 \succ C_1$ is the exception can also be ruled out due to the way we define an exception, namely, that the credential declared worse by the exception is the worst non-empty set of credentials. For example, if the priority vector is $V = [c_2, c_3, c_1]$, then case B-I corresponds to $C_0 = \{c_1\}$ and $C_1 = \{c_3\}$. But no non-empty set C_2 exists s.t. $C_1 \succ C_2$. So $C_2 = \phi$ and therefore $C_0 \nsubseteq C_2$.

The remaining case is A-II where $C_1 \succ C_2$ is the exception. We prove this by contradiction, i.e., assume $C_0 \subseteq C_2$ or to be precise $C_0 \subset C_2$ because $C_0 \neq C_2$. But no non-empty C_0 exists because the only possible C_0 satisfying $C_0 \subset C_2$ is $C_0 = \phi$ and it does not satisfy $C_0 \succ C_1$. This concludes the proof since we ruled out all the four cases. □

Lemma 12. *Given any ID-agnostic tie-breaking function T, the majority judging function J^T_{maj} is well-formed.*

Proof. We prove the three properties. The IA and KR proofs are immediate. We now prove TKR. Given three different credential sets C_0, C_1 and C_2 s.t. $C_0 \succ C_1$, and $C_1 \succ C_2$, we want to show that $C_0 \nsubseteq C_2$ to satisfy TKR.

For any majority-based judging function, if $C \succ C'$ then $|C| \geq |C'|$. So we have $|C_0| \geq |C_1| \geq |C_2|$.

So either $|C_0| > |C_2|$ or $|C_0| = |C_2|$ is true. If $|C_0| > |C_2|$ then $C_0 \nsubseteq C_2$ and we are done.

It remains to prove for $|C_0| = |C_2|$. But in this case $C_0 \nsubseteq C_2$ follows as $C_0 \neq C_2$ (by definition) and $|C_0| = |C_2|$. □

B. Miscellaneous results

Lemma 13. *The profile matrix of $M^{[c_1,c_2]}_{\mathsf{pr}}$ is as per Fig. 1.*

Proof. The 3x3 grid with no safe scenarios are unsuccessful, i.e., won by the attacker due to Lemma 3.

Since the priority judging function is well-formed, $M^{[c_1,c_2]}_{\mathsf{pr}}$ is maximal (Lemma 10). Corollary 1 says that all maximal mechanisms are secure in with-safe-no-stolen scenarios $\Sigma_{\mathsf{wsnt}} = \Sigma_{\mathsf{ws}} \cap \Sigma_{\mathsf{nt}}$ and half of the with-safe-with-stolen scenarios ($\Sigma_{\mathsf{wswst}} = \Sigma_{\mathsf{ws}} \cap \Sigma_{\mathsf{wt}}$).

This fixes the values in all scenarios of the matrix except two: the scenario $\sigma = \{\mathsf{safe}, \mathsf{stolen}\}$ and its complement $\overline{\sigma} = \{\mathsf{stolen}, \mathsf{safe}\}$. Corollary 1 says that at most one of σ, $\overline{\sigma}$ is secure; we need to find out which. The judging function $J^{[c_1,c_2]}_{\mathsf{pr}}$ favors the user in the scenario σ and the attacker in $\overline{\sigma}$. □

Lemma 14. *For all n, the size of all n-credential complete maximal sets is the same.*

Proof. Consider two different complete maximal sets $\mathcal{O}_1$ and $\mathcal{O}_2$. Each mechanism $M_2 \in \mathcal{O}_2$ must be equivalent to or worse from a mechanism $M_1 \in \mathcal{O}_1$ (Definition 13). But all mechanisms in a complete maximal set are maximal, therefore it must be that $M_2 \cong M_1$. So, for each mechanism in $\mathcal{O}_2$, there exists an equivalent mechanism in the other set $\mathcal{O}_1$, and vice versa. And because no two mechanisms within $\mathcal{O}_1$ (or $\mathcal{O}_2$) can be equivalent, there exists a one-to-one mapping between $\mathcal{O}_1$ and $\mathcal{O}_2$. So $|\mathcal{O}_1| = |\mathcal{O}_2|$. □

C. Count of majority judging functions

We now count the number of different majority functions, i.e., tie-breaking functions. A tie occurs when $C_0 \neq C_1$ but $|C_0| = |C_1|$. Let $|C_0| = |C_1| = s$ where $1 \leq s < n$ (note that s can't be equal to n since $C_0 \neq C_1$). For a given s, there are

$$f(n,s) = \frac{\binom{n}{s}\left(\binom{n}{s} - 1\right)}{2},$$

possible tie situations. This is because, there are $\binom{n}{s}$ ways of picking an s-sized set C_0, and $\binom{n}{s} - 1$ ways of picking another set $C_1 \neq C_0$. We divide by two as each tie situation repeats twice. Summing over all s, the total number of distinct tie situations is

$$q(n) = \sum_{s=1}^{n-1} f(n,s). \tag{5}$$

A tie-breaking function can decide each situation in two ways: pick C_0 or C_1. So there are $2^{q(n)}$ different tie-breaking functions, or in other words, $2^{q(n)}$ different majority functions.

$$\begin{aligned}
q(n) &= \sum_{s=1}^{n-1} \frac{\binom{n}{s}\left(\binom{n}{s} - 1\right)}{2} \\
&= \frac{1}{2}\sum_{s=1}^{n-1}\binom{n}{s}^2 - \frac{1}{2}\sum_{s=1}^{n-1}\binom{n}{s} \\
&= \frac{1}{2}\left[\binom{2n}{n} - \binom{n}{0}^2 - \binom{n}{n}^2\right] - \frac{1}{2}\left[2^n - 2\right] \\
&= \frac{1}{2}\left[\binom{2n}{n} - 2\right] - \frac{1}{2}\left[2^n - 2\right] \\
&= \frac{1}{2}\left[\binom{2n}{n} - 2^n\right] \\
&= \binom{2n-1}{n-1} - 2^{n-1}
\end{aligned}$$

Appendix D
Complete Maximal 3-credential Set

A. $\mathcal{O}_{\mathsf{maj},3}$ mechanisms

The twelve majority mechanisms in $\mathcal{O}_3$ are:

Both linear (6 mechanisms): Fix $[c_1, c_2, c_3]$ as the linear priority rule to break ties between 1-credential sets, and use all the six permutations of $[c_1, c_2, c_3]$ to break ties between 2-credential sets.

Both cyclic (2 mechanisms): Two possible cyclic rules exist: clockwise ($c_1 \succ c_2, c_2 \succ c_3, c_3 \succ c_1$) or counter-clockwise. The two mechanisms are: use same or different rules for both 1-credential and 2-credential tie-breakers.

Linear and cyclic (4 mechanisms): Fix $[c_1, c_2, c_3]$ as the linear priority rule to break ties between 1-credential sets and use clockwise or counter-clockwise rule for 2-credential sets to produce 2 mechanisms. Use cyclic rule for 1-credential sets and linear rule for 2-credential sets to obtain two more.

B. Proof of Theorem 3

Before proving that these mechanisms form the complete maximal set for 3 credentials we present a few lemmas. The first one says that if both players use fixed strategies, then a change in the scenario does not change the execution winner.

Lemma 15. *Given two scenarios σ, σ' using the same mechanism M and a user strategy S^U, attacker strategy S^A such that they can be employed in both the scenarios. Then $Win_\sigma(S^U, S^A) = Win_{\sigma'}(S^U, S^A)$.*

Proof. The proof follows straightforwardly because we are using a deterministic automaton and the strategies are deterministic, so irrespective of the scenario, the executions will be the same, and hence the winner. □

The next lemma says that, if the mechanism fails in a scenario σ, then it also fails in all scenarios $\widehat{\sigma}$ that are worse than σ. $\widehat{\sigma}$ is *worse* than σ as the attacker knows (equal or) more credentials whereas the user knows (equal or) fewer credentials. Formally, given a scenario σ, we define a worse scenario $\widehat{\sigma}$ through the following transform:

$$\begin{aligned}\mathsf{safe} &\mapsto \mathsf{safe}/\mathsf{leaked}/\mathsf{lost}/\mathsf{stolen}\\ \mathsf{stolen} &\mapsto \mathsf{stolen}\\ \mathsf{leaked} &\mapsto \mathsf{leaked}/\mathsf{stolen}\\ \mathsf{lost} &\mapsto \mathsf{lost}/\mathsf{stolen}.\end{aligned}$$

Lemma 16. *If a mechanism fails in a scenario σ, then it also fails in a worse scenario $\widehat{\sigma}$, $\forall M$, $\neg Suc(M, \sigma) \implies \neg Suc(M, \widehat{\sigma})$.*

Note that proving the above also implies that success in a worse scenario $\widehat{\sigma}$ implies success in σ.

Proof. Observe that (a) $C^A_\sigma \subseteq C^A_{\widehat{\sigma}}$ and (b) $C^U_{\widehat{\sigma}} \subseteq C^U_\sigma$.

We prove by contradiction. That is, assume that the attacker succeeds in a scenario σ but the user succeeds in the transformed scenario $\widehat{\sigma}$. Let the successful strategy for the user in $\widehat{\sigma}$ be S^U. Since $C^U_{\widehat{\sigma}} \subseteq C^U_\sigma$, the user can employ the strategy S^U in the scenario σ. By our initial assumption, the attacker succeeds in σ. Let the successful attacker strategy be S^A. S^A must win some executions against any user strategy including S^U. So we have $Win_\sigma(S^U, S^A) = \mathrm{A}$.

Since $C^A_\sigma \subseteq C^A_{\widehat{\sigma}}$, the attacker can employ the strategy S^A in the scenario $\widehat{\sigma}$. Any execution in the scenario $\widehat{\sigma}$ where the user employs S^U and the attacker employs S^A results in a win for the user. So we have $Win_{\widehat{\sigma}}(S^U, S^A) = \mathrm{U}$.

But σ and $\widehat{\sigma}$ use the same underlying mechanism M. And if the strategies of the two parties are same, then both $Win_\sigma(S^U, S^A) = \mathrm{A}$ and $Win_{\widehat{\sigma}}(S^U, S^A) = \mathrm{U}$ cannot be true due to Lemma 15 and we arrive at a contradiction. □

We now prove Theorem 3, i.e., that $\mathcal{O}_3$ is made up of the two priority and twelve majority mechanisms only.

Proof. Proving a complete maximal set requires satisfying three requirements per Definition 13:

1) All mechanisms in $\mathcal{O}_3$ must be maximal.
2) All mechanisms in $\mathcal{O}_3$ must be incomparable.
3) No other mechanism exists s.t. it is incomparable to each mechanism in $\mathcal{O}_3$.

The first requirement is met due to Lemma 10, Lemma 11 and Lemma 12. The second one is also met, as can be verified by inspecting the 76 mechanism profiles. (We cross-check this through code.)

We prove the third requirement using a similar approach taken in Theorem 2, but done at a larger scale. We use a constraint solver to show that no 3-credential mechanism M exists satisfying two types of constraints, explained below.

The first type of constraint encodes that M must be incomparable with any mechanism in $\mathcal{O}_3$. Satisfying it requires that for each mechanism $M' \in \mathcal{O}_3$, there exists at least one scenario $\sigma \in prof(M)$ that satisfies $\sigma \notin prof(M')$. This scenario σ must be a with-safe scenario due to Lemma 3. Rephrasing the above, we get $\exists \sigma \in prof(M)$ such that $\sigma \in \Sigma_{\mathsf{ws}} \setminus prof(M')$.

Observe that for any $M' \in \mathcal{O}_3$, $prof(M')$ contains all the with-safe scenarios except some with-safe-with-stolen scenarios. Therefore $\Sigma_{\mathsf{ws}} \setminus prof(M') \subset \Sigma_{\mathsf{wswst}}$. And consequently, $prof(M)$ must include at least one with-safe-with-stolen scenario.

So to prove that no mechanism M exists, it suffices to prove that $prof(M)$ does not contain a with-safe-with-stolen scenario. Let $prof_{\mathsf{wswt}}(M)$ denote the subset of $prof(M)$ containing just the with-safe-with-stolen scenarios. We want to show that $|prof_{\mathsf{wswt}}(M)| = 0$.

We prove this by consider the set of all subsets of the 18 with-safe-with-stolen scenarios satisfying Lemma 2, i.e., no two scenarios in a subset are complements of each other. Denote this set by Δ ($|\Delta| = 3^9$ because the 18 with-safe-with-stolen scenarios can be arranged into 9 rows with each row containing 2 complement scenarios. And we have three ways of making a selection in each row: {none, first scenario, second scenario}). Observe that this is the set of all possible values for $prof_{\mathsf{wswt}}$, i.e., for any mechanism M, it must be that $prof_{\mathsf{wswt}}(M) \in \Delta$.

And we want to find a set in Δ satisfying the two constraints. The first constraint, explained before, is $\forall M' \in \mathcal{O}_3, prof_{\mathsf{wswt}}(M) \cap (\Sigma_{\mathsf{ws}} \setminus prof(M')) \neq \phi$.

The second constraint relies on the following observation. If the mechanism wins in a with-safe-with-stolen scenario σ, Lemma 2 implies that it fails in the complement scenario $\overline{\sigma}$. But then, due to Lemma 16, the mechanism fails in any scenario worse than $\overline{\sigma}$. For example, $prof(M)$ cannot contain the two scenarios $\sigma = \{\mathsf{stolen}, \mathsf{stolen}, \mathsf{safe}\}$ and $\sigma' = \{\mathsf{lost}, \mathsf{safe}, \mathsf{stolen}\}$. This is because the mechanism fails in the complement of σ, $\overline{\sigma} = \{\mathsf{safe}, \mathsf{safe}, \mathsf{stolen}\}$, and σ' is a worse scenario than $\overline{\sigma}$.

The second constraint is $\forall \sigma \in prof_{\mathsf{wswt}}(M)$, $\nexists \sigma' \in prof_{\mathsf{wswt}}(M)$ s.t. the complement $\overline{\sigma}$ is worse than σ'.

The constraint solver[1] outputs the empty set, i.e., no such $prof_{\mathsf{wswt}}(M)$ exists and hence $|prof_{\mathsf{wswt}}(M)| = 0$. This completes the proof that $\mathcal{O}_3$ is complete. □

Appendix E
Applications

We present details of Argent in §E-A and a bank in §E-B. The Paralysis Proofs [14] mechanism is in Fig. 5.

[1]Code at https://pastebin.com/RqjesNZe for anonymous submission.

c_o safe

g_1 \ g_2	St	Le	Lo	Sa
St	1	1	1	1
Le	1	1	1	1
Lo	1	1	1	1
Sa	1	1	1	1

c_o lost

g_1 \ g_2	St	Le	Lo	Sa
St	0	0	0	0
Le	0	0	0	0
Lo	0	0	0	1
Sa	0	0	1	1

c_o leaked

g_1 \ g_2	St	Le	Lo	Sa
St	0	0	0	0
Le	0	0	0	0
Lo	0	0	0	1
Sa	0	0	1	1

c_o stolen

g_1 \ g_2	St	Le	Lo	Sa
St	0	0	0	0
Le	0	0	0	0
Lo	0	0	0	0
Sa	0	0	0	0

TABLE III: Profile of Argent with 2 guardians

A. Argent

1) 2 guardians: We analyze the profile of Argent's mechanism with 2 guardians M^2_{Arg} (Tab. III). Since any mechanism fails in all no-safe scenarios (Lemma 3), we only discuss with-safe scenarios.

c_o safe / stolen: If the owner is safe, the user can revoke all the leaked and stolen guardians (if any), and win. Similarly, if the owner is stolen, the attacker can execute the above strategy and win.

c_o lost: If each of the guardians is either lost or safe, the user wins as the winning user strategy is to use the safe credential to change the lost credentials. Note that a single guardian is enough to change the owner.

In case one of the guardians gets either leaked or stolen, the attacker wins. The attacker's winning strategy is to make the wallet unusable by bringing the automaton to the owner change state C_o (requires one guardian). Even if the user can cancel the owner change (possible in certain scenarios), the attacker can immediately initiate another owner change.

c_o leaked: If both guardians are safe or if one is safe and another is lost, the user wins because she is able to initiate an owner change and the attacker does not know enough credentials to cancel it. Note that starting a guardian change does not help the attacker as the user can cancel it just before the 1.5 day period ends.

If one of the guardians is leaked / stolen, the attacker wins by executing a fast withdrawal.

In summary, the profile of M^2_{Arg} has 22 scenarios in comparison to our maximal mechanisms with 28 scenarios. A more commonly used 2-out-of-3 mechanism's profile has 14 scenarios in comparison (see Tab. V). But M^2_{Arg} is worse than the priority mechanism where owner credential has the highest priority, i.e., $M^2_{\mathsf{Arg}} \prec M^{[c_o,g_1,g_2]}_{\mathsf{pr}}$ (see Tab. IIa).

Multicalls: With multicalls, the mechanism additionally wins in the scenarios: (c_o lost, g_1 leaked, g_2 safe) and (c_o lost, g_1 safe, g_2 leaked). Everything else remains the same.

The winning strategy based on the automaton's state: (a) if it is C_o and the owner change is not started by the user, cancel the current owner change, and immediately start a new owner change, and (b) if the state is anything else, execute an owner change. After the owner becomes safe, we move to an already analyzed scenario where the mechanism succeeds.

Observe that $M^2_{\mathsf{Arg}} \prec M^{[c_o,g_1,g_2]}_{\mathsf{pr}}$ holds even with multicalls.

2) Future extensions: We now discuss some modeling extensions that can enrich the analysis.

Sending money to a trusted address could be treated as win for the user. This can be reasonable when the address belongs to friends / family and their accounts are safe. Note that this is an asymmetric capability that the user alone possesses, i.e., the attacker cannot leverage it. Modeling it is an interesting direction for future work. Exploring larger guardian numbers and attacks that initiate concurrent calls, withdrawing small amounts or changing guardians with overlapping delay periods, would require additional tooling outside the scope of this work.

B. Bank

We illustrate the bank automaton in Fig. 4 and explain its resultant profile (Tab. IV):

If c_m is safe, then irrespective of the state of password, the mechanism succeeds because the user can reset password (if it is unsafe) and then send money to a new account. The above is true even if c_{id} is unsafe because it takes 3 days to change the mobile number, and the user can send all the money to a new account meanwhile. (Note that this is true because we assume that the user knows the credential c_{id}; considering partial knowledge might be more appropriate for this analysis.)

Similarly, if c_m is leaked or stolen (corresponding to phone hijacking attacks like SIM swapping), then the mechanism fails because the attacker can now reset the password and move money.

And if c_m is lost and c_{id} is safe, then the mechanism succeeds because the user can (a) change their mobile number by waiting for 3 days and (b) then reset password (if it is unsafe).

Finally, if c_m is lost and c_{id} is unsafe, the mechanism fails because either the attacker changes the mobile (c_{id} leaked / stolen) or no one does and the account is locked (c_{id} lost).

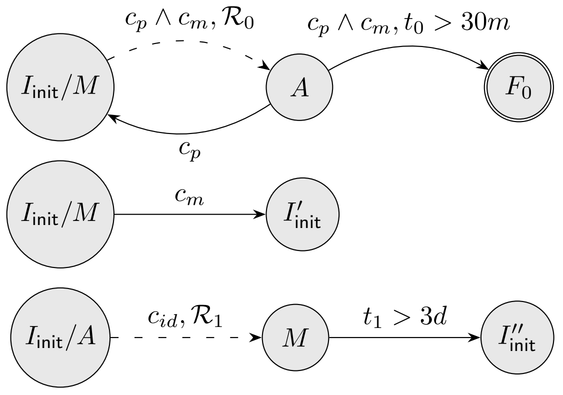


Fig. 4: HDFC bank sub-automata: send money to a new account, change password, and change mobile number respectively.

c_{id} safe

c_m \ c_p	St	Le	Lo	Sa
St	0	0	0	0
Le	0	0	0	0
Lo	1	1	1	1
Sa	1	1	1	1

c_{id} lost

c_m \ c_p	St	Le	Lo	Sa
St	0	0	0	0
Le	0	0	0	0
Lo	0	0	0	0
Sa	1	1	1	1

c_{id} leaked

c_m \ c_p	St	Le	Lo	Sa
St	0	0	0	0
Le	0	0	0	0
Lo	0	0	0	0
Sa	1	1	1	1

c_{id} stolen

c_m \ c_p	St	Le	Lo	Sa
St	0	0	0	0
Le	0	0	0	0
Lo	0	0	0	0
Sa	1	1	1	1

TABLE IV: Profile of the bank mechanism.

c_1 safe

c_2 \ c_3	St	Le	Lo	Sa
St	0	0	0	1
Le	0	0	0	1
Lo	0	0	0	1
Sa	1	1	1	1

c_1 lost

c_2 \ c_3	St	Le	Lo	Sa
St	0	0	0	0
Le	0	0	0	1
Lo	0	0	0	0
Sa	0	1	0	1

c_1 leaked

c_2 \ c_3	St	Le	Lo	Sa
St	0	0	0	0
Le	0	0	0	0
Lo	0	0	0	1
Sa	0	0	1	1

c_1 stolen

c_2 \ c_3	St	Le	Lo	Sa
St	0	0	0	0
Le	0	0	0	0
Lo	0	0	0	0
Sa	0	0	0	1

TABLE V: Profile of the 2-out-of-3 mechanism.

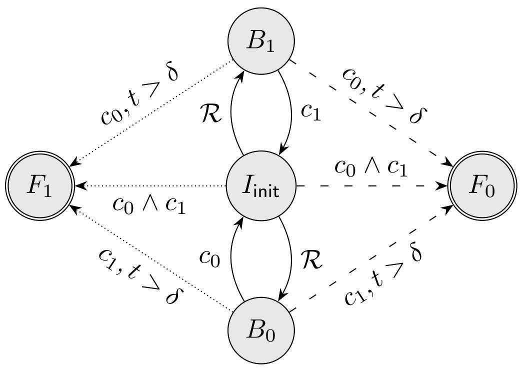


Fig. 5: Paralysis Proofs Mechanism [14] with 2-credentials